

**UNIVERSIDADE FEDERAL DE SERGIPE  
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA  
COMPUTAÇÃO**

**IMPLEMENTAÇÃO DE UMA FOG COMPUTING PARA  
FORNECER StaaS A DISPOSITIVOS IoT UTILIZANDO  
SISTEMAS EMBARCADOS**

**JOSÉ DOS SANTOS MACHADO**

**SÃO CRISTÓVÃO/SE**

**2018**

**UNIVERSIDADE FEDERAL DE SERGIPE  
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA  
COMPUTAÇÃO**

**JOSÉ DOS SANTOS MACHADO**

**IMPLEMENTAÇÃO DE UMA FOG COMPUTING PARA  
FORNECER StaaS A DISPOSITIVOS IoT UTILIZANDO  
SISTEMAS EMBARCADOS**

**Dissertação** apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal do Sergipe (UFS) como parte de requisito para obtenção do título de Mestre em Ciência da Computação.

**Orientador:** Prof. Dr. Edward David Moreno Ordóñez

**SÃO CRISTÓVÃO/SE**

**2018**

**FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL**  
**UNIVERSIDADE FEDERAL DE SERGIPE**

Machado, José dos Santos  
Implementação de uma fog computing para fornecer StaaS a dispositivos IoT utilizando sistemas embarcados / José dos Santos  
M149i Machado ; orientador Edward David Moreno Ordonez. – São Cristóvão, 2018.  
147 f. : il.

Dissertação (mestrado em Ciências da Computação) –  
Universidade Federal de Sergipe, 2018.

1. Ciências da computação. 2. Computação em nuvem. 3. Sistemas embarcados. 4. Internet. I. Ordonez, Edward David Moreno, orient. II. Título.

CDU 004

**JOSÉ DOS SANTOS MACHADO**

**IMPLEMENTAÇÃO DE UMA FOG COMPUTING PARA  
FORNECER StaaS A DISPOSITIVOS IoT UTILIZANDO  
SISTEMAS EMBARCADOS**

**Dissertação** apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal do Sergipe (UFS) como parte de requisitos para obtenção do título de Mestre em Ciência da Computação.

**BANCA EXAMINADORA**

Prof. Dr. Edward David Moreno Ordonez, Presidente  
Universidade Federal de Sergipe (UFS)

Prof. Dr. Admilson de Ribamar Lima Ribeiro  
Universidade Federal de Sergipe (UFS)

Prof. Dr. Ricardo José Paiva de B. Salgueiro  
Universidade Federal de Sergipe (UFS)

Prof. Dr. Hermes Senger  
Universidade Federal de São Carlos (UFCAR)

# **IMPLEMENTAÇÃO DE UMA FOG COMPUTING PARA FORNECER StaaS A DISPOSITIVOS IoT UTILIZANDO SISTEMAS EMBARCADOS**

Este exemplar corresponde à redação na íntegra, da Dissertação de Mestrado, submetida à banca examinadora designada pelo Colegiado do Programa de Pós-Graduação em Ciência da Computação, para o Exame de Defesa do mestrando **JOSÉ DOS SANTOS MACHADO**, como parte de requisitos para obtenção do título de Mestre em Ciência da Computação.

Aprovada em 21 de agosto de 2018

---

Prof. Dr. Edward David Moreno Ordonez (UFS)  
Orientador

---

Prof. Dr. Admilson de Ribamar Lima Ribeiro (UFS)  
Coorientador

---

Prof. Dr. Ricardo José Paiva de B. Salgueiro (UFS)  
Membro

---

Prof. Dr. Hermes Senger (UFSCAR)  
Membro

**À minha esposa Nídia Mary Santos Machado e aos meus filhos Nicole Gabrielle e  
Nícolas Gabriel, pelo amor incondicional.**  
**Aos meus pais, João Batista Machado e Terezinha dos Santos Machado, sempre meu  
porto seguro.**  
**A minha sogra, Juveni do Nascimento Santos, por me amar como um filho.**

**Dedico**

---

## AGRADECIMENTOS

À Deus, por confortar meu espírito nos momentos de tormenta.

Ao Prof. Dr. Edward David Moreno Ordonez, pela dedicação nas orientações neste período de aprendizado, pelo exemplo de profissional.

Ao Prof. Dr. Admilson de Ribamar Lima Ribeiro pela contribuição como coorientador na fase de qualificação.

À minha mulher, esposa e companheira, por sua dedicação e motivação, e por sempre estar ao meu lado na busca de uma construção de uma vida humilde e reta.

Aos meus queridos e amados filhos Nicolás e Nicole, por todo o amor e carinho de sempre.

Aos meus pais, João Batista Machado e Terezinha dos Santos Machado, pelo amor incondicional e sem limites.

Aos meus irmãos pelo apoio sem limites.

Aos meus cunhados e em especial, Gilvan Vieira e Nívia Rocha por me apoiar nos momentos difíceis.

Aos meus colegas de trabalho e em especial, a Raphael Silva Fontes, pelo apoio e discussão crítica do meu trabalho.

Aos professores e colegas do programa de Pós-Graduação em Ciência da Computação, em especial, a Danilo Souza Silva, pelo aprendizado e convivência valorosa.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES, pela oportunidade de qualificação profissional.

Ao Instituto Federal de Sergipe, minha casa profissional, por me permitir mais um avanço na carreira e em especial, a professora Dr<sup>a</sup>. Sônia Pinto de Albuquerque Melo, pela possibilidade de adequação das minhas atividades laborais diárias.

---

“ Deus não escolhe os capacitados, capacita os escolhidos. Fazer ou não fazer algo só depende de nossa vontade e perseverança. ”

“ Eu tentei 99 vezes e falhei, mas na centésima tentativa eu consegui. Nunca desista de seus objetivos mesmo que esses pareçam impossíveis. A próxima tentativa pode ser vitoriosa. ”

Albert Einstein.

---



## LISTA DE FIGURAS

---

1.1	Fog localizada entre borda e nuvem .....	20
2.1	Arquitetura da computação em nuvem .....	27
2.2	Implementação da plataforma OpenStack .....	29
2.3	Arquitetura do servidor OwnCloud .....	30
2.4	Arquitetura Eucalyptus .....	31
2.5	Plataforma OpenNebula .....	32
2.6	Plataforma AbiCloud .....	33
2.7	Plataforma de nuvem Nimbus .....	34
2.8	Componentes da plataforma de nuvem Seafile .....	35
2.9	NextCloud autenticação via LDAP ou Active Directory .....	36
4.1	Cenário para a realização dos testes .....	70
4.2	Cenário para a realização do experimento .....	71
4.3	Raspberry PI 3 Modelo B .....	72
4.4	Banana PI Modelo M3 .....	73
4.5	DELL T410, Banana Pi M3 e Raspberry Pi 3 .....	74
4.6	Benchmarking e ambiente de monitoramento do Smashbox .....	75
4.7	Monitoramento Zabbix .....	76
4.8	Diferença entre VM e Docker .....	77
5.1	Cenário de teste .....	81
5.2	Configuração do arquivo testrun.config .....	82
5.3	Resultado de teste com Smashbox .....	84
5.4	Utilização da CPU dos Equipamentos .....	97
6.1	Fluxograma Web Service FogSys .....	105
6.2	Fluxograma Simulation FogSys .....	107
6.3	IDE PyCharm 2017.3.3 .....	109
6.4	Interface de Login do FogSys .....	109
6.5	Interface de Gerenciamento do FogSys .....	110

6.6	Interface Gerenciamento dos Sensores no FogSys .....	110
6.7	Fluxo dos dados no experimento .....	111
6.8	Cenário do experimento usando FogSys .....	112
6.9	Arquivo de medição .....	114
6.10	Teste de simulação .....	115
6.11	Teste de envio via HTTP .....	115
6.12	Dados no OwnCloud Server .....	116
6.13	Dados no Smartphone .....	117

## LISTA DE GRÁFICOS

---

3.1	Artigos identificados (fase de execução) .....	54
3.2	Resultado da etapa de seleção dos artigos .....	55
5.1	Comparação do test0 - 1 x 1B = 1 Byte .....	87
5.2	Comparação do test1 - 1 x 100Mb = 100 Megabytes .....	90
5.3	Comparação do test2 - 10 x 10Mb = 100 Megabytes .....	93
5.4	Comparação do test3 - 1000 x 10Kb = 10 Megabytes .....	96
5.5	Comparação do test4 - 1 x 500Mb = 500 Megabytes .....	100

## LISTA DE QUADROS

---

2.1	Comparação das plataformas cloud .....	38
3.1	Sumários dos artigos analisados .....	58
4.1	Abreviaturas dos testes .....	69
5.1	Diferentes implementações de sistemas operacionais .....	80

## LISTA DE TABELAS

---

1.1	Características da IoT e Cloud .....	22
2.1	Plataformas de nuvens .....	37
2.2	Comparação de resultados entre a Cloud e a Fog .....	45
2.3	Comparação entre as plataformas embarcadas .....	46
5.1	Resultados do Test0 - 1 x 1B = 1 Byte .....	85
5.2	Resultados do Test1 - 1 x 100Mb = 100 Megabytes .....	88
5.3	Resultados do Test2 - 10 x 10Mb = 100 Megabytes .....	91
5.4	Resultados do Test3 - 1000 x 10Kb = 10 Megabytes .....	94
5.5	Resultados do Test4 - 1 x 500Mb = 500 Megabytes .....	98

## LISTA DE SIGLAS

---

ACM	<i>Association for Computing Machinery</i>
DHCP	<i>Dynamic Host Configuration Protocol</i>
DNS	<i>Domain Name System</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IaaS	<i>Infrastructure as a Service</i>
IDE	<i>Integrated Development Environment</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
IoT	<i>Internet of Things</i>
PaaS	<i>Platform as a Service</i>
QoS	<i>Quality of Service</i>
SaaS	<i>Software as a Service</i>
SDN	<i>Software Defined Networking</i>
StaaS	<i>Storage as a Service</i>
TI	<i>Tecnologia da Informação</i>
TIC	<i>Tecnologia da Informação e Comunicação</i>
VM	<i>Virtual Machine</i>
WSN	<i>Wireless Sensor Network</i>

# SUMÁRIO

---

<b>1 Introdução .....</b>	<b>18</b>
1.1 Motivação .....	21
1.2 Justificativa .....	20
1.3 Hipótese .....	22
1.4 Objetivos da dissertação .....	23
1.5 Metodologia .....	23
1.6 Organização da dissertação .....	24
<b>2 Contextualização Teórica .....</b>	<b>25</b>
2.1 Cloud Computing .....	26
2.1.1 Armazenamento como Serviço (SaaS) .....	27
2.2 Plataformas de Computação em Nuvem .....	28
2.2.1 OpenStack .....	28
2.2.2 OwnCloud .....	29
2.2.3 Eucalyptus .....	30
2.2.4 OpenNebula .....	31
2.2.5 AbiCloud .....	32
2.2.6 Nimbus .....	34
2.2.7 Seafile .....	35
2.2.8 NextCloud .....	36
2.3 Comparando as Plataformas de Nuvens .....	37
2.4 Fog Computing .....	39
2.4.1 Conceito da Fog Computing .....	39
2.4.2 Características da Fog Computing .....	39
2.4.3 Áreas de Aplicação e Atuação da Fog Computing .....	41
2.5 Plataformas para Sistemas Embarcados .....	45
2.6 Considerações Finais do Capítulo .....	48
<b>3 Trabalhos Correlatos .....</b>	<b>49</b>
3.1 Revisão Sistemática .....	50
3.2 Planejamento da Revisão Sistemática .....	51

3.3 Execução da Pesquisa .....	53
3.4 Procedimentos de Seleção e Extração .....	54
3.5 Resultados da Revisão Sistemática .....	56
<b>4 Metodologia para Prototipação e Coleta de Dados .....</b>	<b>65</b>
4.1 Metodologia .....	65
4.1.1 Aplicação da Metodologia .....	66
4.1.2 Projeto e Cenário de Testes .....	68
4.2 Raspberry PI Modelo B .....	71
4.3 Banana PI Modelo M3 .....	72
4.4 Dell PowerEdge T410 .....	73
4.5 Benchmark Smashbox .....	74
4.6 Zabbix .....	76
4.7 Docker Container .....	77
<b>5 Avaliação e Resultados .....</b>	<b>79</b>
5.1 Elaboração do Cenário de Testes .....	79
5.1.1 Configuração do Smashbox .....	82
5.2 Análise e Avaliação dos Resultados .....	83
5.2.1 Test0 0/1/1 .....	84
5.2.2 Test1 0/1/100000000 .....	88
5.2.3 Test2 0/10/10000000 .....	91
5.2.4 Test3 0/1000/10000 .....	94
5.2.5 Test4 0/1/500000000 .....	98
<b>6 Experimento da Fog Computing .....</b>	<b>101</b>
6.1 Sistema FogSys .....	101
6.2 Fluxograma das ações do Sistema FogSys .....	104
6.3 Ambiente de desenvolvimento .....	108
6.4 Experimento da Fog Computing com o Sistema FogSys .....	111
<b>7 Conclusão .....</b>	<b>119</b>
7.1 Artigos publicados .....	120
7.2 Trabalhos futuros .....	121
<b>Referências .....</b>	<b>122</b>
<b>Apêndice A (artigo publicado 1) .....</b>	<b>126</b>
<b>Apêndice B (artigo publicado 2) .....</b>	<b>137</b>



## RESUMO

A IoT (Internet das Coisas) necessita de muitos serviços para ser uma realidade, por ter recurso de processamento e armazenamento muito restrito, a sua interconexão com a *Cloud Computing* é um fator relevante e de elevada atenção para pesquisadores do mundo inteiro. As questões de latência, big data, armazenamento e poder de processamento próximo dos dispositivos de borda, fez com que surgisse a *Fog Computing*. A *Fog Computing* é a interligação entre os dispositivos IoT e a *Cloud Computing*, e pode levar poder de processamento e armazenamento aos dispositivos IoT, sendo possível o surgimento de novas tecnologias e aplicações para poder sanar essas deficiências encontradas hoje. Esta dissertação apresenta o conceito da *Fog Computing*, seus desafios, sua contextualização teórica, os trabalhos correlatos junto com a revisão sistemática e realiza a implementação e análise de uma *Fog Computing*, para fornecer StaaS (*Storage as a Service*), a dispositivos IoT utilizando plataformas de sistemas embarcados e compara seus resultados com os obtidos por um servidor de alto desempenho. Desenvolve o sistema FogSys com o objetivo principal de simular, receber, validar e armazenar os dados de dispositivos IoT para serem transferidos para *Cloud Computing*, funcionando como uma *Fog Computing* para fornecer o serviço de StaaS (*Storage as a Service*). Os resultados demonstraram que a implementação desse serviço em dispositivos de sistemas embarcados pode ser uma boa alternativa para reduzir um desses problemas, no caso, o armazenamento de dados, que atinge hoje os dispositivos IoT.

Descritores: Computação em Nevoeiro, Computação em Nuvem, IoT, OwnCloud, StaaS, Sistemas Embarcados, Raspberry PI, Banana PI, FogSys.

## ABSTRACT

IoT (Internet of Things) needs many services to be a reality because it has a very restricted processing and storage resource, its interconnection with Cloud Computing is a relevant, high attention factor for researchers worldwide. The issues of latency, big data, storage and processing power close to edge devices, led the emergence of Fog Computing. Fog Computing is the interconnection between IoT devices and Cloud Computing, and it can bring processing and storage power to IoT devices, and it enables the emergence of new technologies and applications to solve these shortcomings. This study presents the concept of Fog Computing, its challenges, its theoretical contextualization, the related works together with the systematic review and performs the implementation and analysis of a Fog Computing, to provide StaaS (Storage as a Service), IoT devices using platforms systems and compares their results with those achieved by a high-performance server. It develops the FogSys system with the main objective of simulating, receiving, validating and storing the data of IoT devices to be transferred to Cloud Computing, functioning as a Fog Computing to provide the service of StaaS (Storage as a Service). The results showed that the implementation of this service in devices of embedded systems is a good alternative to reduce one of these problems, in this case, the data storage, which reaches IoT devices today.

**Keywords:** Fog Computing, Cloud Computing, IoT, OwnCloud, StaaS, Embedded Systems, Raspberry PI, Banana PI, FogSys.

### INTRODUÇÃO

A *Cloud computing* e *Internet of Things* (IoT) são duas tecnologias muito diferentes que já fazem parte da nossa vida. Espera-se que sua adoção e uso se tornem cada vez mais difundidos, tornando-os componentes importantes da Internet do Futuro. Um novo paradigma em que *Cloud* e IoT estão ligados em conjunto é o significado que haverá tecnologias avançadas como facilitador de um grande número de cenários de aplicação na vida cotidiana (BOTTA, 2016).

Assim, surge o CloudIoT um paradigma de TI inovador em que *Cloud* e IoT, duas tecnologias complementares, se uniram para se fundir à Internet atual e futura (BABU, 2015).

Nas últimas décadas, observa-se que serviços de computação como armazenamento, processamento de dados e controle foram migrados para a “nuvem”. A oportunidade de computação ilimitada na nuvem permite que os usuários finais acessem amplas informações facilmente, também é possível visualizar que os dispositivos móveis e sensores, como *smartphones*, se tornaram poderosos equipamentos, o que levou ao surgimento de novos sistemas e aplicações.

Estes sistemas e aplicações introduzem novas demandas funcionais em computação e redes que a “nuvem” sozinha não pode atender neste cenário, percebe-se que a computação local em uma borda da rede é muitas vezes necessária. Por exemplo, para processar dados em tempo real, criar contextos de reconhecimento de localização a partir de sensores locais e maximizar a eficiência das comunicações sem fio na borda da rede. No entanto, a nuvem está muito longe dos dispositivos IoT para satisfazer requisitos de latência e é muito centralizada para lidar com a heterogeneidade e diversidade contextual em uma área local, também é muito custoso carregar todos os dados de sensores individuais para a nuvem (STOJMOVIC, 2015).

Para ultrapassar estas limitações, porções da capacidade de computação da nuvem deslocam-se para a borda da rede e formam um ambiente de computação local, isto é, uma *Fog*

*Computing*" chamada também de “nevoeiro”. Ao distribuir os serviços de computação e de rede mais próximos de onde os dados do usuário são gerados, a *Fog Computing* atende melhor às demandas emergentes (MACHADO, 2017).

A *Fog Computing* apresenta uma nova arquitetura que "leva processamento para os dados", enquanto a nuvem "leva os dados para o processamento" (AL-DOGHMAN, 2016). Dessa maneira dispositivos de borda e dispositivos móveis podem estar interligados dentro de uma rede local e executar colaborativamente tarefas de armazenamento, processamento de dados de rede e de controle (HAJIBABA, 2014).

Em uma arquitetura de *Fog Computing*, redes de sensores podem desempenhar um papel significativo na medida em que sensores e atuadores implantados em ambiente distribuído podem vir a ser geradores de dados, controladores para sistemas físicos e plataformas de computação em rede. A *Fog Computing* pode ter *gateways* de borda que têm mais capacidade de computação. Uma rede de sensores, incluindo sensores e atuadores, estará profundamente conectada aos *gateways* de “nevoeiro” e aos dispositivos móveis diretamente e, em seguida, fazer extensas interações com eles, isso fornecerá novos serviços que conectam o ambiente físico à infraestrutura cibernética.

A *Fog Computing* pode vir a resolver muitos problemas da Internet das Coisas (IoT), por exemplo, os serviços da *Fog* serão capazes de melhorar a largura de banda e as restrições de custo das comunicações de longo alcance. No entanto, muitos desafios ainda permanecem na computação em *Fog*, como modelar uma arquitetura de sistema para a *Fog*; como implementar, organizar e gerenciar dispositivos da *Fog*; como a *Fog* interage com a nuvem e com os dispositivos; e como gerenciar a conectividade física e lógica da *Fog*.

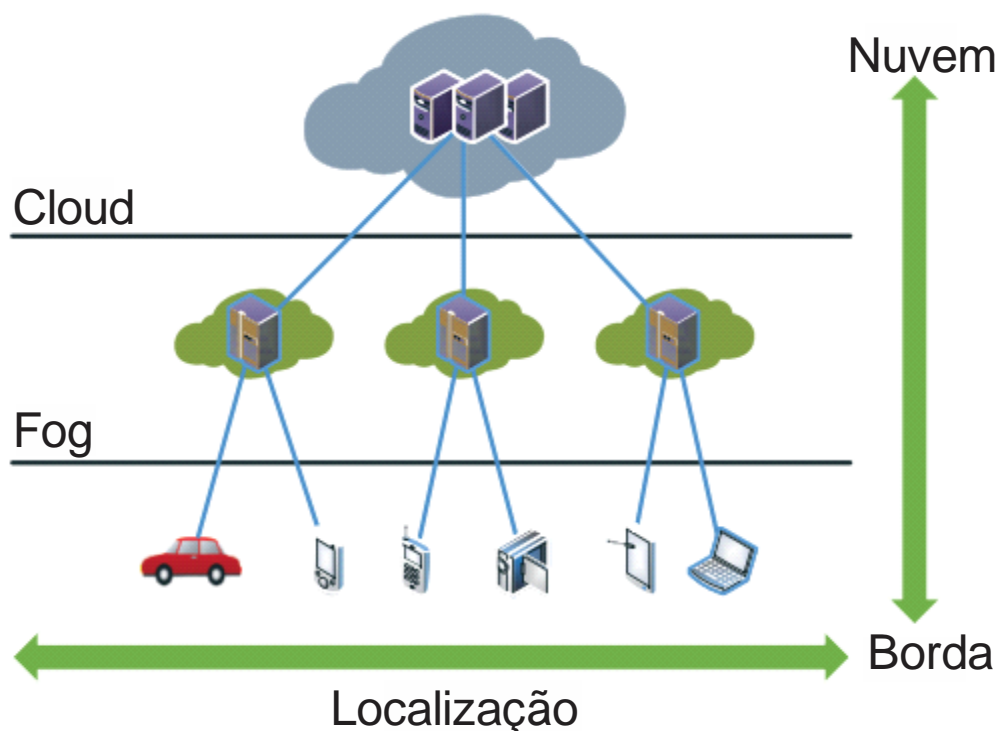
## 1.1 Motivação

A *Fog Computing* é uma extensão da clássica *Cloud Computing* para a borda da rede (como a névoa é uma nuvem perto do chão). Ela foi projetada para suportar aplicações da Internet das coisas (IoT) caracterizadas por limitações de latência, exigência de mobilidade e de distribuição geográfica. Apesar da computação, armazenamento e rede serem recursos tanto da *Cloud* e da *Fog*, o último tem características específicas, localização na borda e reconhecimento do local implicando baixa latência, distribuição geográfica e um grande número de nós em contraste com a nuvem centralizada, apoio à mobilidade (através de acesso sem fio), interação em tempo real (em vez de processos *batch*) e suporte para interação com a *Cloud* (FAN; JAIN, 2016).

BOTTA *et al.* (2016), fazem uma análise que mostra como a construção de projetos de *Fog Computing* é um desafio, de fato, a adoção de abordagens baseadas em *Fog* requer vários algoritmos e metodologias relacionadas com a confiabilidade das redes de dispositivos inteligentes, e que operam sob condições específicas que pedem técnicas de tolerância a falhas específicas.

A *Fog Computing* é proposta para resolver os problemas acima mencionados. Como a *Fog Computing* é implementada na borda da rede, ela fornece baixa latência, definição geográfica de localização e melhora a qualidade de serviço (QoS) para aplicações que usam *streaming* e aplicações em tempo real (KUMAR, 2016). Exemplos típicos incluem automação industrial, transporte, redes de sensores e atuadores. Além disso, essa nova infraestrutura suporta heterogeneidade, pois os dispositivos *Fog* incluem dispositivos para usuários finais, pontos de acesso, roteadores e *switches*. Conforme se afirma em STOJMENOVIC (2015), o paradigma *Fog* está bem posicionado para análise de dados em tempo real, suporta pontos de coleta de dados densamente distribuídos e oferece vantagens em entretenimento, publicidade, computação pessoal e outras aplicações. A figura 1.1 mostra a localização entre *Fog Computing* e *Cloud Computing*.

Fig. 1.1 - Fog localizada entre borda e nuvem



Fonte: Adaptado de STOJMENOVIC (2016)

## 1.2 Justificativa

Internet das Coisas (IoT) é uma das tendências crescentes em Tecnologias de Informação e Comunicação (TIC), sendo caracterizada pela presença de sensores e atuadores, dispersos, necessitando de plataformas de hospedagem, geralmente disponíveis em várias ordens de magnitude, possivelmente heterogêneos ao longo de vários eixos, por exemplo, arquitetura, conjunto de instruções, sistema operacional e tempo de execução (LONGO, 2015). Já existem algumas aplicações na vida social cotidiana, tais como monitoramento ambiental, transporte inteligente, sistema de monitoramento de saúde, casas inteligentes, etc. (BABU, 2015; BOTTA; DÍAZ, 2016).

IoT representa uma interligação de um grande número de objetos inteligentes com baixos recursos através da WSN (rede de sensores sem fios), que dispõe de uma grande rede de sensores e atuadores, com restrita capacidade de processamento e memória (BABU, 2015). Necessitando de recursos de segurança eficiente para garantir confiabilidade e privacidade, esses são importantes desafios para sua popularização (BABU, 2015; DÍAZ, 2016). Hoje em dia, sistemas embarcados estão ficando cada vez mais poderosos e flexíveis, em termos de comportamento reprogramável e facilidade de uso, para avançar esta evolução neste cenário ambicioso e abrangente, é necessário adotar e usar tecnologias adequadas (BOTTA, 2016; LONGO, 2015).

Como a Internet das coisas tem capacidades limitadas de poder de processamento e de armazenamento, também as questões como desempenho, segurança, confiabilidade e privacidade, a *Fog Computing* pode ser a solução para esses problemas. A integração da Internet das coisas com *Cloud* é necessária para prover serviços que são limitados, como armazenamento e poder de processamento. *Cloud* pode até se beneficiar com a Internet das coisas, que pode estender seus limites com as coisas do mundo real de forma mais dinâmica e distribuída e, entregar grande número de serviços em tempo real (BABU, 2015). As características da IoT e *Cloud* de diferentes paradigmas são traçadas na tabela 1.1, *Cloud* atuará como camada intermediária entre as aplicações e as coisas para esconder todas as funcionalidades e complexidades necessárias para processar posteriormente. A tabela 1.1 mostra o desenvolvimento de aplicações futuras, onde a informação, processo de coleta e transmissão vai oferecer novos desafios a enfrentar em um ambiente de nuvem.

Tabela 1.1 - Características da IoT e Cloud

<b>IoT</b>	<b>Cloud</b>
IoT é Pervasiva (Coisas colocadas em todos os lugares)	A nuvem é ubíqua (Recursos utilizáveis de todos os lugares)
Estas são coisas do mundo real	Trata-se de recursos virtuais
Tem capacidade limitado de computação	É praticamente ilimitada a capacidade computacional
Possui armazenamento limitado ou sem recursos de armazenamento	A nuvem tem recursos de armazenamento praticamente ilimitados
Este usa a Internet como ponto de convergência	Este usa a Internet para a prestação de serviços
Esta é uma grande fonte de dados	Significa gerenciar grande quantidade de dados

Fonte: Adaptado de BABU (2015)

Desta maneira percebe-se que atualmente a Integração da *Cloud* com IoT (CloudIoT), é um grande desafio para pesquisadores, e a implementação de uma *Fog Computing* com baixo recurso, utilizando equipamentos simples e acessíveis como uma plataforma de sistema embarcado Raspberry PI, em vez de potentes e caros servidores, pode ser uma boa alternativa para suprir as limitações da IoT, como poder de processamento, armazenamento e recursos de rede.

### 1.3 Hipótese

Para nortear o desenvolvimento desta pesquisa e, ainda, para atingir os objetivos da proposta deste trabalho, algumas hipóteses foram levantadas:

Q1 - Qual aplicação de *Cloud Computing* se adequa para fornecer *Storage as a Service* (StaaS) e é suportada pelas plataformas embarcadas Raspberry PI e Banana PI, para servir com *Fog Computing*?

H1 – A primeira hipótese desta dissertação é que a aplicação Owncloud é adequada para fornecer *Storage as a Service* (StaaS), para dispositivos IoT, funcionando como uma *Fog Computing*.

Q2 – O desempenho obtido pela implementação de uma *Cloud Storage* nas plataformas de sistemas embarcados Raspberry PI e Banana PI é satisfatório, quando comparado com implementação em computadores de alto poder de processamento?

H2 - O desempenho obtido pela implementação de uma *Cloud Storage* nas plataformas de sistemas embarcados Raspberry PI e Banana PI é menor, porém pode ser considerado satisfatório.

## 1.4 Objetivos da Dissertação

O objetivo principal desta dissertação de mestrado é propor a implementação e análise de uma *Fog Computing*, para fornecer StaaS (*Storage as a Service*), a dispositivos IoT utilizando plataformas de sistemas embarcados.

Para atingir esse objetivo principal, foram elencados objetivos específicos, a saber:

- Identificar qual plataforma de nuvem é adequada para fornecer StaaS (*Storage as a Service*) e que seja possível sua implementação numa plataforma de sistemas embarcados como Raspberry PI e Banana PI;
- Implementar a plataforma de nuvem em dispositivos de sistemas embarcados baseados em Raspberry PI e Banana PI, obedecendo o conceito e características de uma *Fog Computing*;
- Homologar a qualidade do serviço disponibilizado, no caso StaaS (*Storage as a Service*) para dispositivos IoT, através de técnica de avaliação ou *benchmark* específico para StaaS (*Storage as a Service*).
- Desenvolver um sistema WEB para armazenar e simular o envio de dados dos dispositivos IoT através de *Upload* e sincronizar com a *Cloud Storage Client* em *real time*, fazendo o papel de uma *Fog Computing*.

## 1.5 Metodologia

Este projeto de dissertação de mestrado está caracterizado como uma pesquisa aplicada que contém objetivos exploratórios e descritivos, com abordagem qualitativa com caráter bibliográfico e experimental. Inicialmente foi feita uma atualização do estado da arte, um estudo aprofundado das plataformas existentes de *Cloud Computing*, para escolher a melhor opção a ser implantada para dispositivos com recursos limitados como é o caso do Raspberry PI. Após as devidas análises, foi realizada a implementação de uma *Fog Computing* usando sistemas embarcados nas plataformas de dispositivos Raspberry PI e Banana PI, seguindo o método de



ANTONI (2015) e CHEN (2017), como referência. Após a instalação são realizados testes de desempenho e experimento do serviço implementado StaaS (*Storage as a Service*), para comprovar a eficiência da implantação da *Fog Computing*, visando fornecer StaaS (*Storage as a Service*) para diversos dispositivos IoT.

## 1.6 Organização da dissertação

O texto desta dissertação está organizado em sete capítulos que fornecem toda a base conceitual, metodológica e empírica para o entendimento completo do estudo realizado. Os tópicos a seguir descrevem o conteúdo de cada um destes capítulos.

- O capítulo 1 apresenta as argumentações, problemática e hipóteses sobre o tema, os objetivos, as definições preliminares de literatura;
- O capítulo 2 expõe a contextualização teórica da literatura com a revisão de literatura relacionada a sistemas aplicados a *Fog Computing*, e a integração da *Cloud Computing* à IoT (*Internet of Things*), apresentação e comparação das plataformas de *Cloud Computing*, apresentação e comparação das plataformas de sistemas embarcados;
- O capítulo 3 demonstra os trabalhos correlatos com a revisão de literatura adotada (Revisão Sistemática), bem como são apresentados os resultados dessa revisão e a síntese do processo de sumarização dos trabalhos estudados;
- O capítulo 4 apresenta o método utilizado para a prototipação e coleta dos dados obtidos para análise nos dispositivos implementados, tais como: metodologia utilizada, arquitetura de simulação ou cenário de testes, dispositivos de sistemas embarcados Raspberry Pi 3, Banana Pi M3 e o servidor DELL PowerEdge T410;
- O capítulo 5 apresenta a análise e avaliação com o *benchmark* Smashbox para validar o desempenho do serviço de armazenamento StaaS (*Storage as a Service*);
- O capítulo 6 apresenta um experimento de implementação simulando um ambiente de *Fog Computing*, utilizando o sistema FogSys;
- Por fim, o capítulo 7 apresenta as conclusões desta dissertação e considerações finais, suas contribuições, artigos publicados e trabalhos futuros.

### CONTEXTUALIZAÇÃO TEÓRICA

A Internet tem testemunhado duas mudanças radicais na última década, o crescimento rápido da computação em nuvem e os dispositivos IoT. Motivado por estas duas tendências uma infinidade de pesquisas tem sido conduzida para suportar a computação em nuvem, que liga as nuvens e os dispositivos IoT, alavancando tanto a poderosa capacidade de computação da nuvem quanto o suporte à mobilidade de dispositivos móveis e IoT.

Grande parte dos trabalhos tem focado em como efetivamente descarregar tarefas computacionais intensivas de dispositivos com recursos limitados, para a nuvem e obter os resultados desejados (CHEN, 2017).

No entanto, devido à latência de rede frequentemente imprevisível, especialmente em um ambiente móvel, muitas vezes a computação em nuvem não pode atender aos requisitos rigorosos de latência dos aplicativos, segurança e privacidade em área restrita geograficamente (ZHU, 2013). Por outro lado, a crescente quantidade de dados gerados por dispositivos e sistemas com poucos recursos pode se tornar impraticável para transportar dados através de redes para nuvens remotas (OSANAIYE, 2017).

Para isso, surgiu um novo paradigma, a *Fog Computing*. A *Fog Computing* é a computação em nuvem que distribuirá com serviços avançados de computação, armazenamento, rede e gerenciamento mais próximos dos usuários finais, enviando informações dos dispositivos IoT para *Cloud Computing*, formando assim uma plataforma distribuída e virtualizada, assim, também é referido como computação de borda.

A *Fog Computing* oferece muitas vantagens desejadas pelas aplicações de hoje, como processamento em tempo real, rápida escalabilidade, compartilhamento de recursos e armazenamentos locais. Como tal, a *Fog Computing* rapidamente conquistou muita atenção da indústria e da academia. É naturalmente a ponte entre a Internet das Coisas (IoT) e a computação em nuvem, com a infraestrutura de computação existente na Internet (LI, 2017).

As aplicações atuais e futuras que exigem a *Fog Computing* incluem veículos conectados, veículos de piloto automático, redes inteligentes, redes de sensores e atuadores sem fio, casas inteligentes, cidades inteligentes, sistemas conectados de petróleo e gás e sistemas móveis de saúde (CHEN, 2017).

## 2.1 Cloud Computing

A *Cloud Computing* é uma modalidade de computação recente, que vem se tornando popular desde meados de 2007. Essa modalidade de computação permite que o usuário acesse uma grande quantidade de informação e serviços que não estão armazenados localmente no seu dispositivo. *Cloud Computing* é considerada o próximo passo na evolução natural dos sistemas computacionais (ANTONI, 2015).

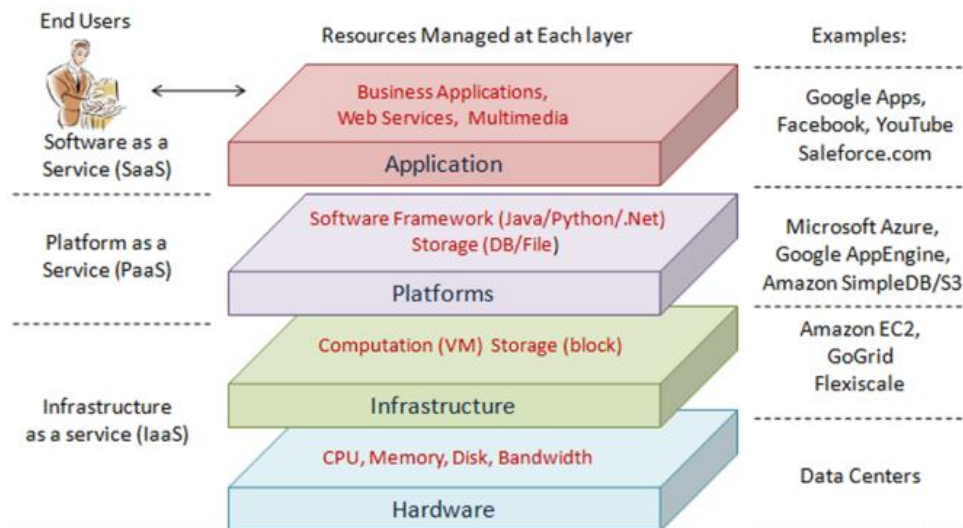
A *Cloud Computing* é uma forma de computação baseada na Internet que pode fornecer recursos de *hardware* e *software*, de acordo com as necessidades. *Cloud Computing* virtualiza a infraestrutura de computação, redes, armazenamento e outros, para formar uma grande gama de recursos compartilhados, mensurável e dinâmico, e fornece um modelo de computação para todos os tipos de usuários na forma de serviços controlados através de uma plataforma de gestão (ANTONI; BABU; HUO, 2015; BOTTA; DÍAZ, 2016; ZHANG, 2010).

Os principais modelos de serviço são: (i) SaaS - (*Software as a Service*); (ii) - PaaS (*Platform as a Service*) e (iii) - IaaS (*Infrastructure as a Service*) (ANTONI, 2015; ZHANG, 2010).

- (i) SaaS - (*Software as a Service*): Refere-se ao fornecimento de aplicações sob demanda através da Internet. Exemplos de fornecedores de SaaS incluem Salesforce.com, Rackspace e SAP Business ByDesign.
- (ii) PaaS (*Platform as a Service*): Refere-se ao fornecimento de recursos da camada de plataforma, incluindo suporte ao sistema operacional e de desenvolvimento de *software*. Exemplos de provedores de PaaS incluem o Google App Engine, Microsoft Windows Azure e Force.com.
- (iii) IaaS (*Infrastructure as a Service*): Refere-se ao provisionamento sob demanda de recursos de infraestrutura, geralmente em termos de VMs. O proprietário da

nuvem que oferece IaaS é chamado um provedor de IaaS. Exemplos de fornecedores de IaaS incluem Amazon EC2, GoGrid e Flexiscale.

Fig. 2.1 - Arquitetura da computação em nuvem



Fonte: ZHANG (2010)

De um modo geral, a arquitetura de um ambiente de computação em nuvem pode ser dividida em 4 camadas: A camada de *Hardware / Datacenter*; a camada da Infraestrutura; a camada de Plataforma e a camada de Aplicação, como mostrado na figura 2.1.

### 2.1.1 Armazenamento como Serviço (StaaS)

A computação em nuvem e, em particular, os serviços de armazenamento em nuvem tornaram-se uma parte cada vez mais importante do setor de tecnologia da informação nos últimos tempos. O número de opções de armazenamento em nuvem está aumentando, com a maioria dos fornecedores fornecendo uma quantidade variada de armazenamento livre antes de cobrar por níveis de armazenamento maiores, devido ao número crescente desses serviços disponíveis, muitos pesquisadores usaram a frase *Storage as a Service* (StaaS), como uma extensão do *Software as a Service*, para descrever esse tipo de serviço (MARTINI, 2013).

## 2.2 Plataformas de Computação em Nuvem

A Computação em nuvem possibilita acessar recursos computacionais (por exemplo, servidores, armazenamento, redes, serviços e aplicações) de maneira prática e sob demanda, rapidamente, e que podem ser liberados para o usuário sem qualquer envolvimento gerencial. Isso pode ser muito importante para agilizar o desenvolvimento do trabalho, reduzir custos, facilitar o emprego de recursos de alto processamento, evitar gastos com manutenção e licenças de *software*.

A dificuldade não está somente em implementar uma nuvem, mas também em escolher a ferramenta mais apropriada para o projeto de redes. Nesta dissertação de mestrado, o escopo são as ferramentas *open source* para administração de nuvem que suportam o modelo IaaS, PaaS, SaaS e StaaS. Existem várias plataformas disponíveis para implementação a exemplo da OpenStack, OwnCloud, Eucalyptus, OpenNebula, Abicloud, Nimbus, SeaFile e NextCloud, que são comentadas em seguida.

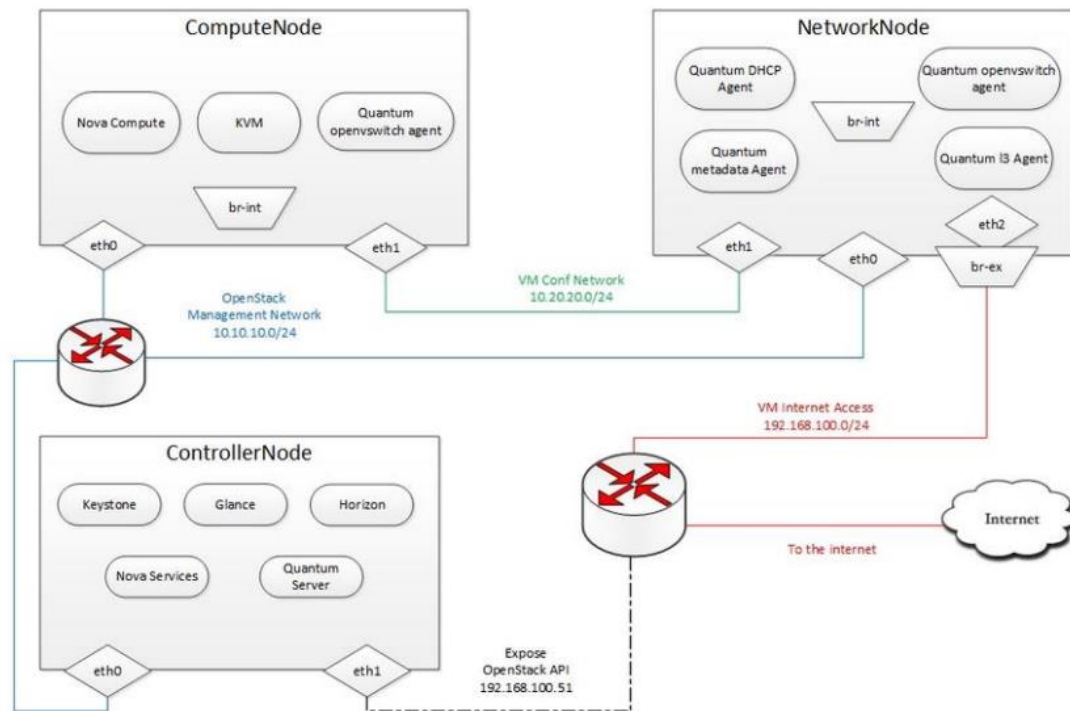
### 2.2.1 OpenStack

OpenStack é uma plataforma de *Cloud Computing* que foi desenvolvida pelo provedor de hospedagem Rackspace e NASA, para ajudar os provedores de serviços de nuvem e empresa a construir serviços de infraestrutura de nuvem. O projeto OpenStack foi pensado como um sistema operacional em nuvem, assim qualquer organização ou indivíduo pode construir seu próprio ambiente de computação em nuvem (IaaS) com base em OpenStack (HUO, 2015).

OpenStack adota concepção modular, seus módulos principais são Nova (serviço de computação), Swift (serviço de armazenamento de objetos) e Glance (serviço de imagem). O serviço de computação é uma unidade de plataforma de nuvem que fornece ferramentas de gestão para gerir a execução, redes e usuários da instância de controle. O serviço de armazenamento de objetos é um sistema escalável que suporta uma variedade de aplicações, tais como replicação de dados e arquivamento, vídeo ou imagem e outros tipos de serviços de acesso a dados em massa. E o serviço de imagem é um sistema de armazenamento de imagens, consulta e gerenciamento de máquina virtual (VM) (HUO, 2015).

Na Figura 2.2, três servidores, incluindo um nó de controle, um nó de rede e um nó de computação foram implantados em um ambiente de laboratório, do qual o Swift foi implantado no nó de computação (HUO, 2015).

Fig. 2.2 - Implementação da plataforma OpenStack



Fonte: HUO (2015)

### 2.2.2 OwnCloud

OwnCloud é uma aplicação WEB escrita usando a linguagem PHP. Sua primeira versão foi lançada em 2012 com o propósito de fazer sincronização de arquivos entre vários dispositivos (ANTONI, 2015). Tal característica é útil quando se quer compartilhar arquivos com outros usuários e até mesmo fazer *backups* dos dados. As principais vantagens de se utilizar essa ferramenta são: gratuita e código fonte é *open source*, permitindo que o usuário altere o código conforme sua necessidade e possibilita a criação de uma nuvem de armazenamento privada sendo uma alternativa às nuvens públicas (PRINCY, 2015).

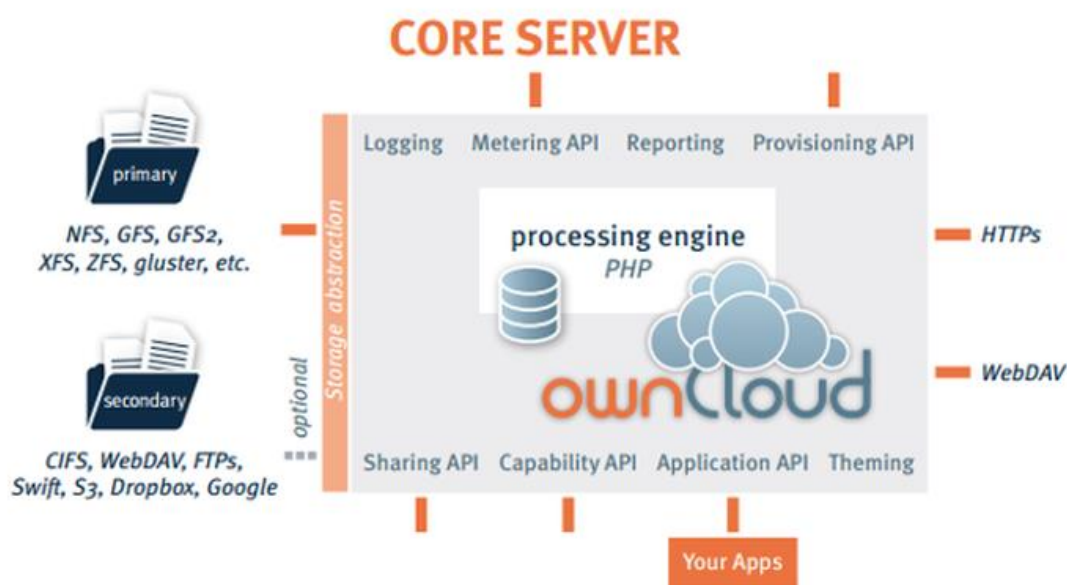
A plataforma é dividida em duas partes; Cliente e Servidor. (i) O Cliente é a parte responsável por decidir quando os dados devem ser enviados e fazer o *download* dos arquivos atualizados do servidor. O cliente está disponível para as versões *desktop* (Linux, Mac e Windows) e *mobile* (Android, BlackBerry e IOS). Ainda é possível acessar os arquivos diretamente do browser através de uma interface WEB. (ii) O Servidor é o ponto de armazenamento centralizado dos dados sendo responsável por enviar os novos arquivos para todos os clientes relacionados àquela conta de usuário. Por ser uma aplicação escrita em PHP, é necessário ter o interpretador PHP instalado no servidor. Além disso é necessário ter um

gerenciador de banco de dados para armazenamento dos metadados dos arquivos e um servidor WEB onde a instância do OwnCloud estará disponível (ANTONI, 2015).

A escalabilidade do OwnCloud é um fator relevante quando se fala em computação na nuvem. Há relatos de organizações que usam o OwnCloud que chegam a atender 500 mil usuários e trabalham com vários terabytes, porém para atingir essa escalabilidade são necessárias configurações adicionais que não estão definidas na instalação padrão (ANTONI, 2015).

A figura 2.3, mostra uma visão da arquitetura do servidor OwnCloud e sua forma de comunicação com suas aplicações.

Fig. 2.3 - Arquitetura do servidor OwnCloud



Fonte: <https://owncloud.com/>

### 2.2.3 Eucalyptus

A Eucalyptus (*Elastic Utility Computing Architecture*), projeto conceitualmente iniciado na UCSB - Universidade da Califórnia Santa Babara (USA), especificamente como um código aberto usado para construir plataforma de nuvem privada. Atualmente passou por modificações e hoje é administrada pela companhia Eucalyptus (NWOBODO, 2015).

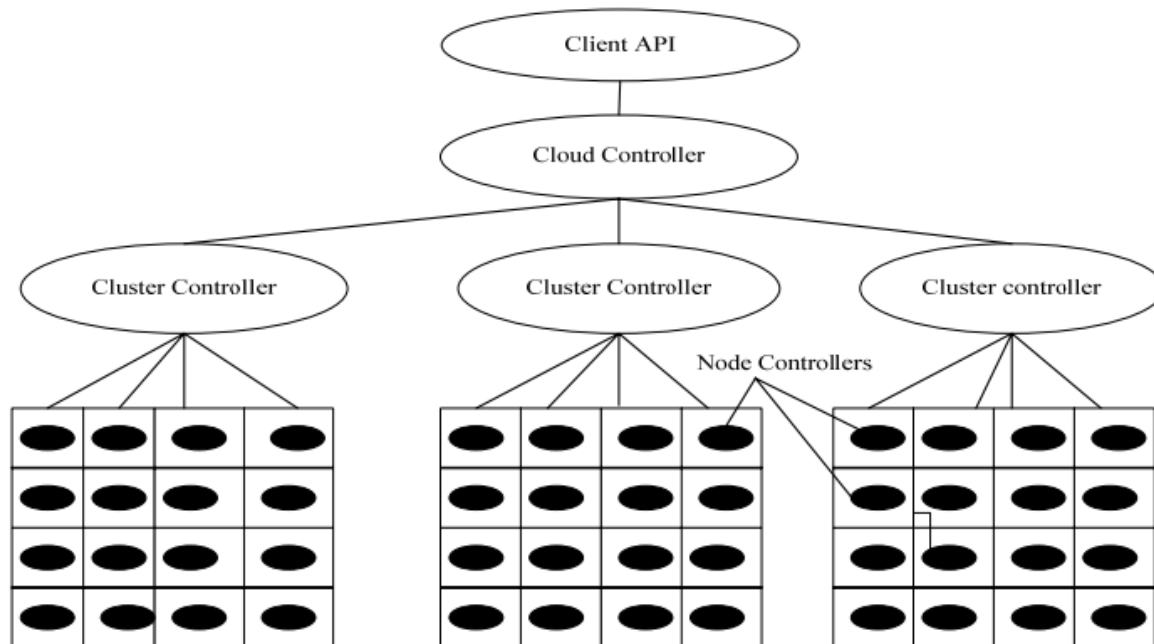
A plataforma é uma implementação de código aberto do Amazon EC2 & S3 usada para construir *Cloud* privada e híbrida compatível com AWS, APIs e com modificação mínima ou

extensão também pode suportar outros clientes. Eucalyptus é uma plataforma projetada que atende às necessidades comerciais da nuvem EC2. Ela consiste em um programa chamado *euca2ools*, um *front end* para usuários, semelhante ao programa EC2 (NWOBODO, 2015).

Eucalyptus é estruturado como computação *Elastic* que permite a conexão do programa de usuário para o sistema principal. A tecnologia é uma infraestrutura que usa estação de trabalho e *clusters* para implementar o *elastic*, utilizado na computação em nuvem e bem conhecido em nível de condição de serviço, baseado em computação padrão que permitem aos usuários melhorar sua capacidade de rede e aumentar seu poder de processamento.

A figura 2.4, mostra uma visão da arquitetura da plataforma de nuvem Eucalyptus e seus *cluster* de controles.

Fig. 2.4 - Arquitetura Eucalyptus



Fonte: NWOBODO (2015)

## 2.2.4 OpenNebula

OpenNebula é uma tecnologia de nuvem importante que evoluiu após muitos anos de pesquisa consistente, e é um projeto desenvolvido pela União Europeia para dar eficiência às máquinas virtuais, escalabilidade e gerenciamento em nível maciço de infraestrutura distribuída (NWOBODO, 2015). A OpenNebula é uma ferramenta de computação em nuvem de código aberto, especificamente projetada como modular para permitir fácil integração com diferentes



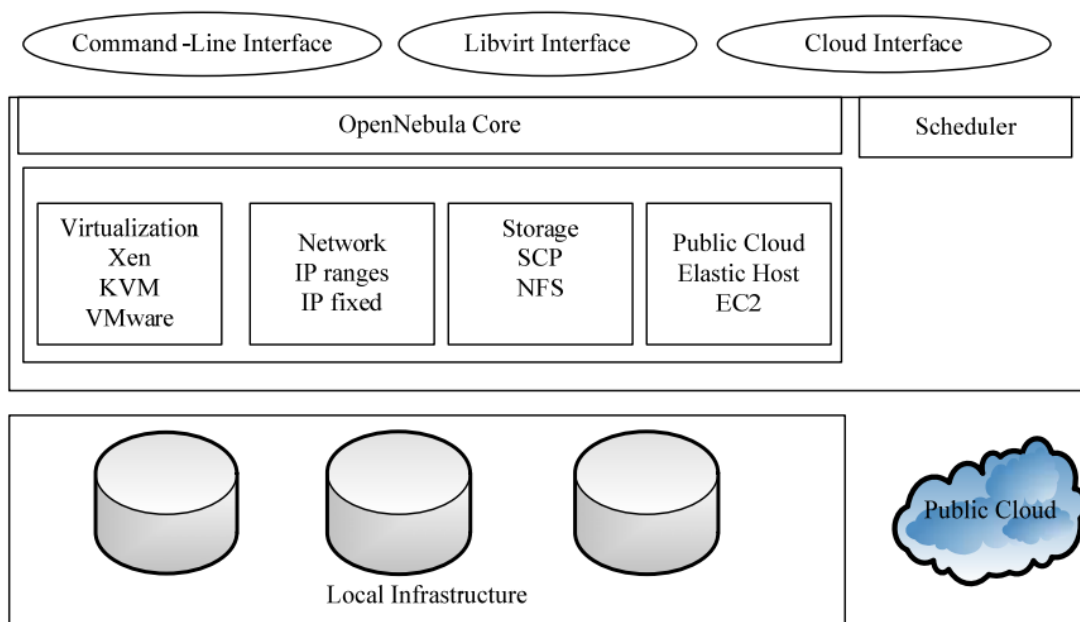
ambientes e *hypervisors*, gerenciar a heterogeneidade e a complexidade da grande infraestrutura distribuída.

OpenNebula é um mecanismo flexível, escalável e aberto de infraestrutura virtual, capaz de permitir a sincronização de redes, armazenamento e técnicas virtuais. Isso pode permitir aos usuários implementar dinamicamente serviços em infraestrutura distribuída com base em estratégias de *datacentres* e recursos remoto de nuvem.

Além disso, a OpenNebula facilita a evolução da nuvem, permitindo aos usuários integrar e alavancar a infraestrutura de TI existente, fornecendo uma camada de gerenciamento flexível, abrangente, aberta e extensível, capaz de orquestrar e automatizar a operação em nuvem empresarial, além da solução de implantação para armazenamento, virtualização e monitorização (NWOBODO, 2015).

A plataforma consiste em três componentes principais como mostra a figura 2.5 abaixo: *Core Virtual Infrastructure Manager*, *Capacity Manager* e *Drivers*.

Fig. 2.5 - Plataforma OpenNebula



Fonte: NWOBODO (2015)

### 2.2.5 AbiCloud

A plataforma AbiCloud de computação foi projetada e desenvolvida pela Abiquo, uma empresa de computação em nuvem estabelecida em Barcelona Espanha, especializada no

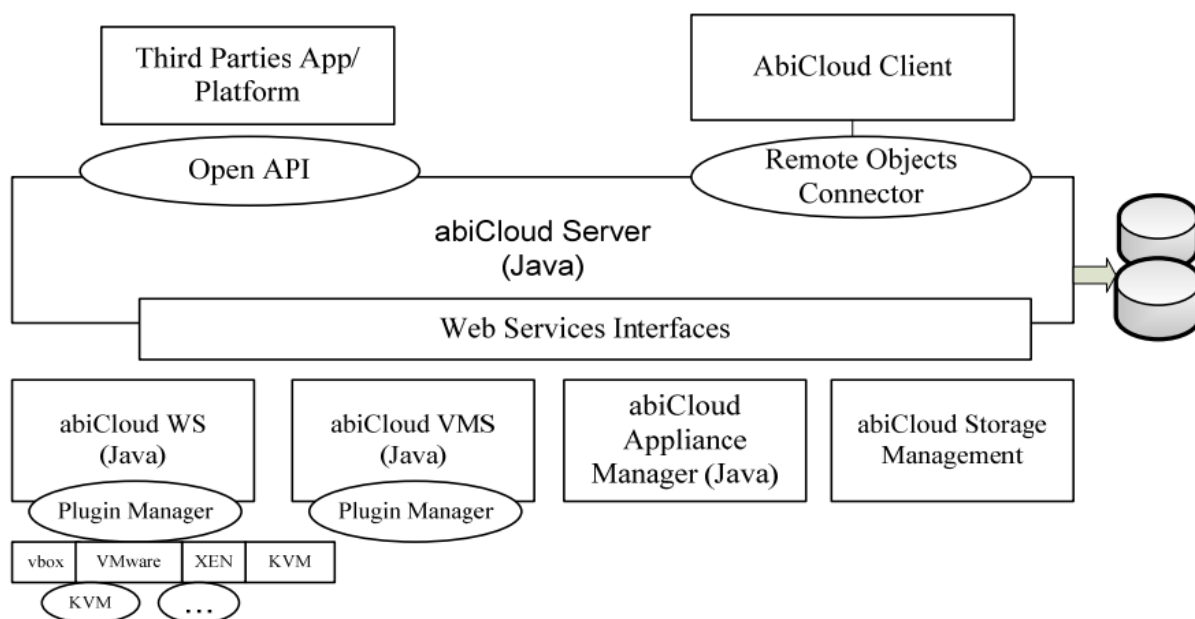
desenvolvimento de plataforma de nuvem. NWOBODO *et al.* (2015), afirmam que a inovação computacional da AbiCloud pode ser usada para construir, gerenciar e integrar um ambiente homogêneo, numa infraestrutura privada e pública virtualizada em nuvem.

Além disso, a plataforma pode permitir fácil escalonamento automático de usuários, orquestração e a implantação de vários utilitários em nuvem, incluindo o gerenciamento de sistemas de armazenamento, servidores, dispositivos virtuais, redes e aplicativos. Esta tem a potencialidade de realizar serviços de IaaS em um curto período de tempo, e alavancar mais essa capacidade na prestação de serviços de hospedagem gerenciada ou autoatendimento IaaS via interface simplificada ou a combinação de ambos (NWOBODO, 2015).

AbiCloud tem interface de gerenciamento baseada na WEB, constitui uma classificação excepcional em comparação com outras plataformas de nuvem, principalmente concebida para implementar serviços usando processo de linha de comando. Os usuários têm a capacidade de implementar novos serviços, como máquinas virtuais (VM), por “*drag and drop*” (arrastar e soltar), nomenclatura utilizada nas interfaces gráficas de computadores, é a ação de clicar em um objeto virtual e “arrastá-lo” a uma posição diferente ou sobre um outro objeto virtual. Assim que a implantação for concluída, os usuários podem usar a API RESTful, bem como outra integração para conectar todos os seus serviços, tais como: sistema de faturamento, CMDB, portal de cliente, sistema de *backup*, sistema de OSS ou BSS.

A figura 2.6, mostra uma visão da arquitetura da plataforma de nuvem AbiCloud e a interligação entre os seus componentes.

Fig. 2.6 - Plataforma AbiCloud



Fonte: NWOBODO (2015)

### 2.2.6 Nimbus

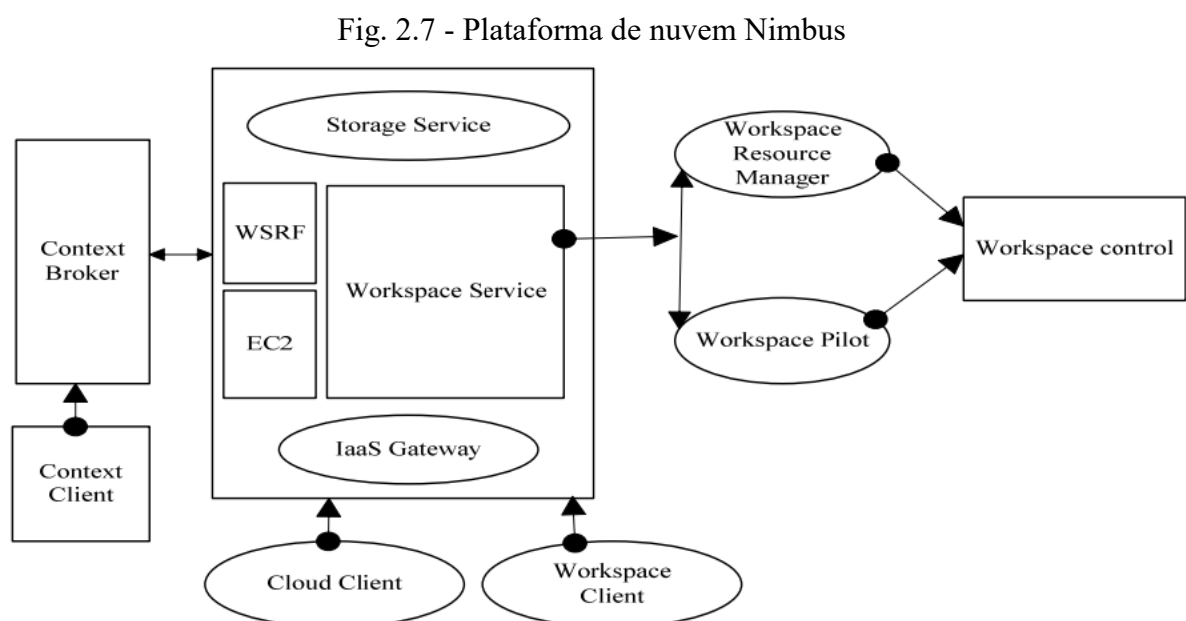
A plataforma de nuvem Nimbus é uma ferramenta integrada aberta, usada para fornecer infraestrutura como serviço para uma solução de computação em nuvem, desenvolvida pela Universidade da Flórida e Chicago (NWOBODO, 2015). A plataforma Nimbus foi especificamente projetada com base no interesse da comunidade científica, como agendadores de lote, credenciais de proxy e alocação de melhores esforços, etc.

Embora tenha recentemente incentivado aplicações e dados não científicos, ela permite que os usuários forneçam e construam vários recursos de computação remota sob demanda através da implantação de máquinas virtuais, Nimbus permite a combinação da Amazon, OpenStack e várias outras nuvens (NWOBODO, 2015).

A nuvem Nimbus parece ser mais complicada do que muitas plataformas de nuvens, pelo motivo de que é necessário inserir funcionalidade usando linhas de comandos para poder obter resultado, fazendo Nimbus parecer complexa. No entanto, o *software* é particularmente único com base em maior nível de flexibilidade, uma vez que suporta quase todos os hypervisores.

A possibilidade de um *backup* consistente e regular da carga de trabalho da plataforma, ajuda a evitar problemas de perda de dados em caso de mau funcionamento. A Nimbus é fornecida sob licença da versão 2 do Apache, como a OpenNebula, a plataforma Nimbus suporta as tecnologias de virtualização XEN e KVM, exceto VMware (NWOBODO, 2015).

A figura 2.7, mostra uma visão da arquitetura da plataforma de nuvem Nimbus e seus principais componentes de APIs.



Fonte: NWOBODO (2015)

### 2.2.7 Seafile

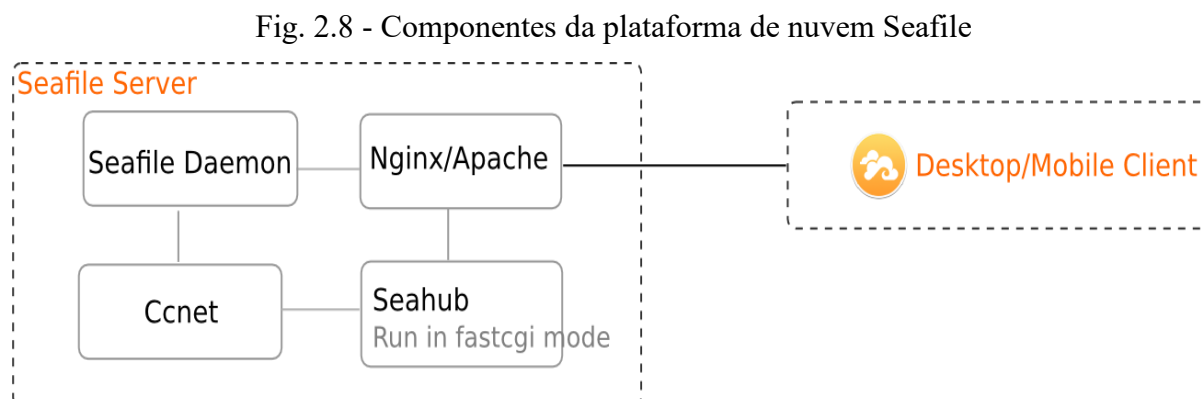
O Seafile é um sistema de armazenamento em nuvem de código aberto com criptografia de arquivos e compartilhamento de grupo. Coleções de arquivos são chamadas de bibliotecas, e cada biblioteca pode ser sincronizada separadamente. Uma biblioteca pode ser criptografada com uma senha escolhida pelo usuário. Esta senha não está armazenada no servidor, portanto, mesmo o administrador do servidor não pode visualizar os conteúdos de um arquivo.

O Seafile permite aos usuários criar grupos com sincronização de arquivos, um wiki e discussões para permitir uma colaboração fácil em torno de documentos dentro de uma equipe.

O servidor Seafile consiste nos seguintes componentes:

- **Seahub** (django): O *frontend* da WEB. O pacote do servidor Seafile contém um leve servidor gunicorn de HTTP em Python, que serve como servidor de website. Por padrão, o Seahub é executado como um aplicativo dentro do gunicorn. Também pode configurar o Seahub para executar no modo *fast-cgi*, através do servidor Nginx ou do Apache. Isso é recomendado para configurações de produção.
- **Servidor Seafile** (seaf-server): Responsável pelo serviço de dados, lida com carregamento de arquivos, *download* e sincronização. O servidor Seafile por padrão escuta na porta 8082. Podendo configurar o servidor Nginx ou Apache para tráfego de *proxy* para a porta local 8082.
- **Servidor Ccnet** (ccnet-server): Responsável pelo serviço RPC para permitir a comunicação entre vários componentes. O Ccnet é usado apenas para comunicação interna.

A figura 2.8, mostra como os clientes do Seafile acessam arquivos quando configurado o Seafile através do servidor Nginx ou Apache.



Fonte: <https://manual.seafile.com>

## 2.2.8 NextCloud

NextCloud é um conjunto de *software* cliente-servidor para fornecer serviços de armazenamento de arquivos em nuvem. É funcionalmente semelhante ao Dropbox, embora o NextCloud seja gratuito e de código aberto, permitindo que qualquer um instale e opere em um servidor privado. Em contraste com os serviços proprietários, como o Dropbox, a arquitetura aberta permite adicionar funcionalidades ao servidor na forma de aplicativos. NextCloud é uma customização da conhecida plataforma de nuvem de armazenamento OwnCloud.

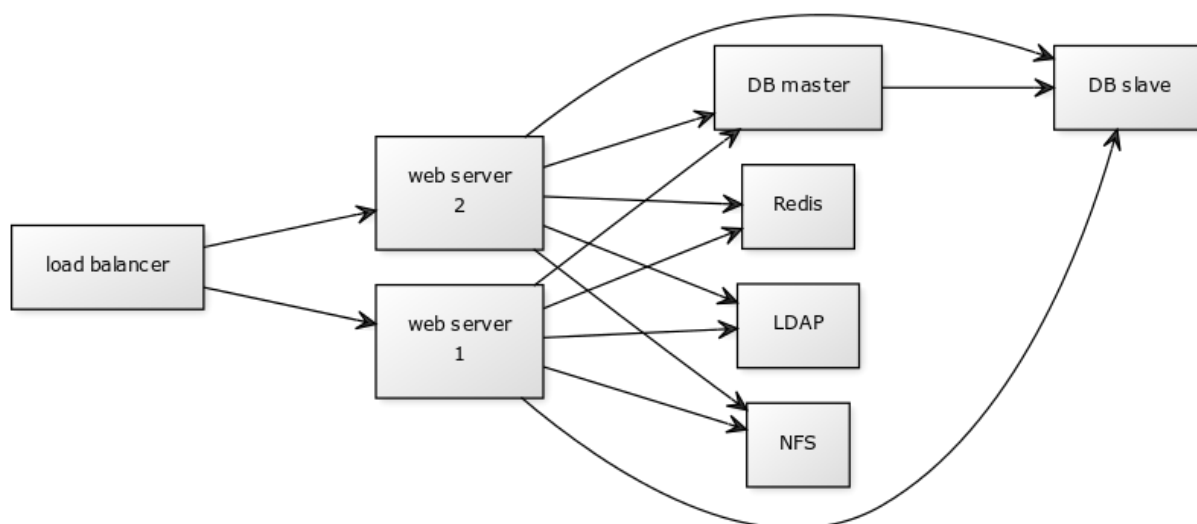
É uma solução flexível de armazenamento, sincronização e compartilhamento de arquivos, sendo todos os sistemas de códigos abertos. É composto pelo servidor NextCloud, que é executado em sistema operacional Linux, aplicativos de clientes para Microsoft Windows, Mac OS X e Linux, e clientes móveis para sistemas operacionais Android e Apple iOS.

O servidor Nextcloud está disponível:

- Como um servidor gratuito, com todas as funcionalidades disponível para comunidade, com todos os recursos corporativos.
- Ou com suporte empresarial completo, incluindo acesso de telefone e e-mail aos desenvolvedores do NextCloud. Sendo essa opção não gratuita.

As distribuições dos sistemas operacionais que oferecem uma maneira mais segura de instalar os vários componentes são as versões atualizadas do Linux. É recomendado o *Red Hat* Enterprise Linux 7 ou o Ubuntu 16.04, pois ambos têm suportes comerciais que podem ser adquiridos. Debian e Ubuntu são gratuitos, e incluem pacotes de *software* mais recentes. O CentOS é o clone do Red Hat Enterprise Linux, com suporte para a comunidade. A figura 2.9, mostra uma implementação da plataforma NextCloud.

Fig. 2.9 - NextCloud autenticação via LDAP ou Active Directory



<https://docs.nextcloud.com>

## 2.3 Comparando as Plataformas de Nuvens

NWOBODO *et al.* (2015), os autores apresentam uma comparação de quatro das plataformas aqui apresentadas (Eucalyptus, AbiCloud, OpenNebula e Nimbus), porém não constam as plataformas (OpenStack, OwnCloud, Seafile e NexCloud), nem a comparação de características importantes para essa pesquisa como da memória mínima requerida para sua implementação, a tabela 2.1 apresenta essa comparação.

Tabela 2.1 - Plataformas de nuvens

	<b>Eucalyptus</b>	<b>OPenNebula</b>	<b>Abicloud</b>	<b>Nimbus</b>
Cloud type	Public	Private	Public/private	Private
Scalability	Scalable	Dynamical, scalable	Scalable	Dynamical, scalable
Cloud form	IaaS	IaaS	IaaS	IaaS
Compatibility	Support EC2, S3	Open, multi- platform	Not supported EC2	Support EC2
Deployment	Dynamical deployment	Dynamical deployment	Pack and redeploy	Dynamical deployment
Deployment manner	Commandline	Commandline	Web interface drag	Commandline
Transplant ability	Common	Common	Easy	Common
Hypervisors support	VMware, Xen, KVM	Xen, VMware, KVM	Virtualbox, Xen, VMware, KVM	Xen, KVM
Web interface	Web service	Libvirt, EC2, OCCL, API	Libvirt	EC2 WSDL WSRF
Structure	Module	Module	Open platform, encapsulate core	Lightweight components
Reliability	--	Rollback host and VM	--	---
OS support	Linux	Linux	Linux	Linux
Development language	java	java	Ruby, C++, python	Java, python
Security	Public/Private key authentication	Authen.password, RSA, SSH, LDAP,	Code Access System(CAS)	PKI
VM build	unavailable	unavailable	Drag & drop	unavailable
OS Licence	BSD Licence	Apache version 2 licence	Common Public Attribution Licence	Apache version 2 licensee

Fonte: NWOBODO (2015)

Atualmente existe uma grande quantidade de plataformas de computação em nuvem com diferentes implementações, características e vantagens, a fim de ter uma melhor compreensão, o quadro 2.1 apresenta uma comparação das plataformas aqui apresentadas (Eucalyptus, OpenNebula, AbiCloud, Nimbus, OpenStack, OwnCloud, Seafile e NextCloud). Sendo que cinco plataformas oferecem IaaS (Infraestrutura como Serviço) e três SaaS (*Storage as a Service*), que são as três últimas mencionadas, existem outras plataformas que fornecem SaaS, é o caso das plataformas Resilio, Syncany, Syncthing, Sparkleshare, Pydio e Cozy, porém este

trabalho fará uma comparação das plataformas antes mencionadas. E alguns parâmetros utilizados na análise incluem, tipo de plataforma, suporte a hypervisores, serviço disponibilizado, SO suportado, linguagem de desenvolvimento e memória requerida.

Quadro 2.1 - Comparação das plataformas cloud

	Tipo da Nuvem	Hypervisor Suportado	Serviço Nuvem	SO Suportado	Linguagem Desenvolvida	Memória Requerida
<b>OpenStack</b>	Pública e Privada	VMware, KVM, Xen e Hyper-V	IaaS	Linux	Python	8 GB e 4 GB RAM
<b>Eucalyptus</b>	Pública	VMware, Xen e KVM	IaaS	Linux	Java	4 GB RAM
<b>AbiCloud</b>	Pública e Privada	VirtualBox, Xen, VMware e KVM	IaaS	Linux	Ruby, C++ e Python	2 GB RAM
<b>OpenNebula</b>	Privada	VMware, Xen e KVM	IaaS	Linux	Java	2 GB RAM
<b>Nimbus</b>	Privada	Xen e KVM	IaaS	Linux	Java e Python	2 GB RAM
<b>OwnCloud</b>	Privada	—	StaaS	Linux	PHP	512 MB RAM
<b>Seafile</b>	Privada	—	StaaS	Linux	Python	512 MB RAM
<b>NextCloud</b>	Privada	—	StaaS	Linux	PHP	512 MB RAM

Fonte: Própria do autor (2017)

De acordo com o quadro 2.1, embora a maioria das plataformas sejam diferentes na implementação de suporte a *hypervisores*, exceto as OwnCloud, Seafile e NextCloud que não suportam, todas executam sobre o sistema operacional Linux, observa-se que todas têm suporte a IaaS (Infraestrutura como Serviço), exceto as OwnCloud, Seafile e NextCloud que suportam StaaS (*Storage as a Service*). Observa-se também que as plataformas foram desenvolvidas usando-se de linguagens diferentes (Python, PHP, Java, Ruby e C++).

Porém, o mais importante para ser analisado no quadro 2.1 é a característica referente ao mínimo requerido de memória RAM para instalação de cada plataforma. Pelo motivo da proposta desta dissertação ser a implementação de uma plataforma de nuvem em plataforma de sistemas embarcados como o Raspberry PI, percebe-se que as melhores opções das plataformas aqui apresentadas para sua implementação são, sem dúvida, as OwnCloud, Seafile e NextCloud,

pelo fato de requerer a menor quantidade de memória RAM 512 MB, já que o Raspberry PI só tem disponível 1 GB de RAM e todas as outras plataformas superam essa capacidade. Neste trabalho foi decidido implementar e testar uma dessas plataformas, a OwnCloud, pelo motivo do sistema de *benchmark* Smashbox suportar essa plataforma e ser uma das mais utilizadas no meio científico.

## 2.4 Fog Computing

A Internet das Coisas (IoT) só resume a uma tecnologia de ponta para perceber o mundo real ao nosso redor, através das redes de computadores, e iniciou uma transição proeminente na forma de como a comunicação e as interações do nosso mundo podem acontecer (ALRAWAIS, 2017). A forte necessidade de remediar as crescentes preocupações com relação ao enorme fluxo de dados em tempo real e a utilização dos limites de largura de banda disponível, levou ao nascimento da *Fog Computing*, que intensa, mas não exclusivamente, opera ao longo da borda da rede (AL-DOGHMAN, 2016).

### 2.4.1 Conceito da Fog Computing

O conceito da *Fog Computing* foi adotado pela *Cisco Systems* como um novo paradigma em 2012 (HAJIBABA, 2014). A *Fog Computing* é um paradigma inovador que realiza computação distribuída, serviços de rede e armazenamento, além da comunicação entre os *Data Centers* da *Cloud Computing* até os dispositivos ao longo da borda da rede (OSANAIYE; MACHADO, 2017). Essa comunicação amplia as operações e serviços inerentes à computação em nuvem, permitindo assim uma nova geração de aplicativos. A principal função é filtrar e agregar dados para os centros de dados da *Cloud* e aplicar inteligência lógica a dispositivos finais.

### 2.4.2 Características da Fog Computing

A *Fog Computing* é semelhante à computação em nuvem em muitos aspectos, no entanto, pode ser diferenciado do anterior pelo fato de estar próximo dos dispositivos finais, a distribuição espacial geograficamente menor e a possibilidade de apoio à mobilidade



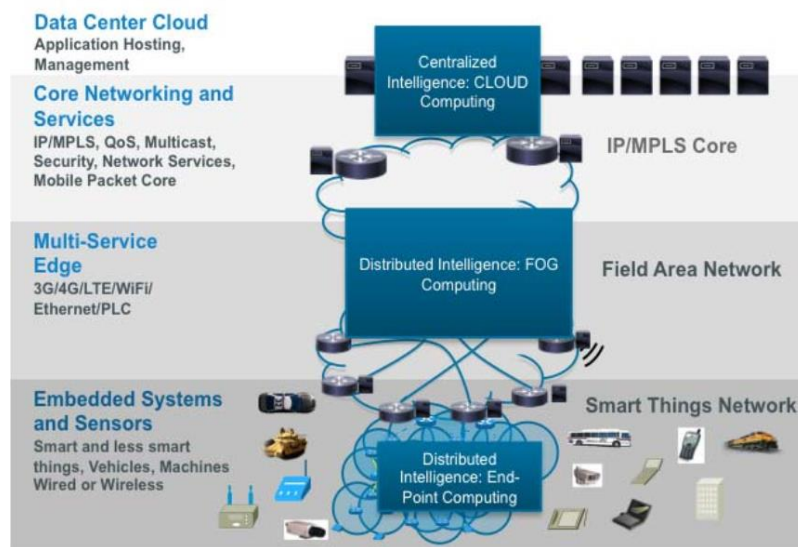
(CHIANG, 2017). Como o processamento baseado em *Fog* ocorre ao longo da borda da rede, os resultados finais refletem uma percepção de localização altamente melhorada, baixa latência e Qualidade de Serviço (QoS), em aplicações de *streaming* e tempo real, os nós de nevoeiro são dispositivos heterogêneos, que vão desde servidores, pontos de acesso, roteadores de borda até dispositivos finais como telefones celulares, relógios inteligentes, sensores e etc. (MACHADO, 2017).

As principais características da *Fog Computing* são:

- **Heterogeneidade:** A *Fog Computing* é uma plataforma virtualizada que oferece serviços computacionais, de rede e de armazenamento entre a computação em nuvem e dispositivos finais de diferentes tipos e formas.
- **Distribuição geográfica:** A *Fog Computing* possui uma implementação amplamente distribuída para oferecer serviços de alta qualidade para dispositivos finais móveis e fixos.
- **Localização de borda, percepção de localização e baixa latência:** O conceito da *Fog Computing* foi implementado para suprir a falta de suporte para pontos finais, com serviços de qualidade à beira da rede.
- **Interação em tempo real:** Várias aplicações de *Fog*, como sistemas de monitoramento de tráfego, exigem processamento em tempo real em vez de processamento em lote.
- **Suporte à mobilidade:** O suporte à mobilidade é essencial para muitos aplicativos de *Fog Computing*, para permitir a comunicação direta com dispositivos móveis usando protocolos como o protocolo de separação de localização / ID da Cisco, que desacopla a identidade do host da identidade de localização usando um sistema de diretório distribuído.
- **Grande escala de redes de sensores:** Isso é aplicável ao monitorar o ambiente ou em rede inteligente, usando sistemas inerentemente distribuídos que requerem computação distribuída ou recursos de armazenamento.
- **Prevalente para acesso sem fio:** A maioria dos pontos de acesso sem fio e o *gateway* de dispositivos móveis são exemplos típicos de um nó de *Fog* na rede.
- **Interoperabilidade:** Os componentes de *Fog* devem ser capazes de interoperar para garantir suporte para ampla gama de serviços, como transmissão de dados.

A Internet das Coisas (IoT) são os dispositivos inteligentes conectados que tiveram um crescimento exponencial em termos de tecnologias, participação no mercado e aprovação dos consumidores, pavimentando o caminho para a evolução dos princípios da *Fog Computing*, melhorando gradualmente oportunidades produtivas em domínios como redes de veículos e o *Smart Grid*, as vantagens deste procedimento de computação para serviços em vários domínios são necessários e devem ser investigados (MACHADO, 2017).

Fig. 2.10 - Arquitetura da Fog Computing



Fonte: ZHU (2013)

*Fog Computing* permite maior suporte e melhor tempo de resposta à Internet das Coisas, é adequada para solicitações de serviço em tempo real, e para compartilhar recursos de forma eficiente, uma eficiente e cooperativa utilidade baseada em estratégia de emparelhamento é necessária entre os nós do IoT (STEINER, 2016). A figura 2.10 apresenta arquitetura da *Fog Computing* com a sua localização.

### 2.4.3 Áreas de Aplicação e Atuação da Fog Computing

A *Fog* funciona como um *link* entre IoT e a Nuvem para introduzir as funcionalidades extras necessárias para o processamento específico da aplicação, como filtragem e agregação. Antes de transferir os dados para a Nuvem, deve ser capaz de decidir o que deve ser enviado (o conteúdo), como (formato de dados) e quando enviar (tempo). Durante esse processo, ela

também precisa excluir alguns dados redundantes ou inválidos, e agregar os dados complementares no espaço e no tempo dimensionado (AL-DOGHMAN, 2016).

Diferentes aplicações de *Fog Computing* foram sugeridas na literatura. Segundo OSANAIYE *et al.* (2017), as categorias das aplicações de *Fog Computing* são divididas em 3: (i) Aplicações em tempo real; (ii) Aplicações quase em tempo real; (iii) Aplicações introduzidas em redes.

- (i) As aplicações em tempo real são aplicações de baixa latência, que funcionam dentro de um período de tempo pré-definido, sendo classificada pelo usuário como imediata ou urgente.
- (ii) Quase em tempo real, por outro lado, são aplicações que estão sujeitas a atraso de tempo introduzido pelo processamento de dados ou transmissão de rede, entre o momento em que ocorre um evento e o uso dos dados para processamento.
- (iii) A *Fog Computing* também pode ser introduzida em uma rede (para aplicativos que não necessite de processamento e transmissão em tempo real), para reduzir a quantidade de tráfego no núcleo de processamento.

Vamos apresentar algumas áreas de aplicação e atuação da *Fog Computing*, disponibilizada na literatura e que poderá ser beneficiada com a sua implementação.

#### **a) Vídeo streaming**

A transmissão de aplicativos e serviços de vídeo aproveitará alguns dos principais benefícios da *Fog Computing*, que incluem conhecimento de localização, baixa latência, suporte para mobilidade e análises em tempo real. Vários dispositivos inteligentes suportam uma aplicação de vigilância inteligente, que pode ser usada por agentes da lei para exibir fluxos de eventos em vídeos de suspeitos em tempo real.

#### **b) Jogos**

O advento da computação em nuvem forneceu uma plataforma para jogos de computador sem usuários (jogadores) preocupados com os requisitos de *hardware*. Os provedores de jogos em nuvem nos últimos tempos têm expandido rapidamente a infraestrutura da nuvem, para fornecer o serviço de jogo sob demanda (GoD) aos usuários através da Internet. É oferecido

remotamente, permitindo um jogo interativo que pode ser acessado e decodificado por dispositivos finais, como *smartphones* ou *tablets*.

#### **c) Cuidados de saúde**

As aplicações IoT forneceram uma abordagem estruturada para melhorar nossos serviços de saúde. Isso é conseguido através da implantação de sistemas de monitoramento onipresentes e transmissão dos dados para dispositivos de *Fog* em tempo real, antes de enviar a informação para a nuvem para posterior análise e diagnóstico.

#### **d) Sistema inteligente de semáforo (STLS)**

O semáforo inteligente foi idealizado por veículos inteligentes conectados e sistema de transporte avançado. Ele interage localmente com uma série de nós sensores para detectar a presença de ciclistas, motociclistas ou pedestres e também estima a velocidade e a distância dos veículos que se aproximam. Esta informação pode ser usada para prevenir acidentes ao enviar sinais de aviso prévio aos veículos que se aproximam.

#### **e) Cidades inteligentes**

Uma cidade inteligente é a chave essencial para aplicação IoT, que varia desde o gerenciamento inteligente de tráfego, até o gerenciamento de energia de edifícios. O conceito de cidade inteligente atraiu o grande interesse dos setores de ciência da engenharia, da comunidade científica e de profissionais da área, como um meio para superar os desafios associados ao rápido crescimento urbano.

#### **f) Veículos inteligentes**

O advento da computação em nuvem móvel exige o estudo de seus agentes, como veículos, robôs e seres humanos, que interagem juntos para detectar o meio ambiente, processar os dados e transmitir os resultados. Veículo conectado através da *Fog Computing* poderá se comunicar com seu ambiente interno e externo.

#### **g) Smart Grid**

Os aplicativos de balanceamento de carga de energia podem ser executados em dispositivos de borda de rede, como medidores inteligentes e micro grades. Com base na

demanda de energia, na disponibilidade e no preço mais baixo, esses dispositivos alternam automaticamente para energias alternativas, como a energia solar e o vento.

#### **h) Redes de sensores e atuadores sem fio**

As redes de sensores sem fio tradicionais são insuficientes em aplicações que vão além de detecção e rastreamento, que exigem que os atuadores exerçam ações físicas como abrir, fechar ou mesmo movimento de sensores. Nesse cenário, os atuadores que atuam como dispositivos de *Fog* podem controlar o próprio processo de medição, a estabilidade e os comportamentos oscilatórios criando um sistema de circuito fechado.

#### **i) Controle de construção inteligente**

As aplicações deste cenário são facilitadas por sensores sem fio implantados para medir temperatura, umidade ou níveis de vários gases na atmosfera de construção. Neste caso, as informações podem ser trocadas entre todos os sensores em um piso, e suas leituras podem ser combinadas para formar medições confiáveis. Os sensores irão usar a tomada de decisão e a ativação de dispositivos de *Fog* para reagir com as informações dos dados recebido.

#### **j) IoT e sistemas ciberfísicos (CPSs)**

Os sistemas baseados em *Fog Computing* estão se tornando uma classe importante de IoT e CPSs. Com base nas operadoras de informação tradicionais, incluindo Internet e rede de telecomunicações, a IoT é uma rede que pode interconectar objetos físicos comuns com endereços identificados. Os CPSs apresentam uma combinação dos elementos computacionais e físicos do sistema.

#### **k) Redes Definidas por Software (SDN)**

A estrutura da *Fog Computing* pode ser aplicada para implementar o conceito SDN para redes de veículos. SDN é um paradigma emergente de computação em rede e tornou-se um dos tópicos mais populares na indústria de TI. Ela separa as camadas de controle e comunicação de dados. O controle é realizado em um servidor centralizado e os nós seguem o caminho de comunicação decidido pelo servidor.

#### **l) Trem autossustentável**

Automanutenção de trens é uma importante área de aplicação. As variações de temperatura podem ser detectadas pelo sensor de monitoramento de rolamento de esferas no

trem e podem notificar automaticamente qualquer tipo de transtorno. Isso ajudará na prevenção de desastres.

Devido à sua recente introdução e emergência, não há nenhuma arquitetura padrão disponível em relação ao gerenciamento de recursos baseado em *Fog*, e por esse motivo ainda existe um modelo simples para esse propósito, considerando a previsão de recursos, a alocação de recursos e os custos de forma realista e dinâmica, considerando também o tipo, e características dos clientes, este modelo é adaptável às exigências de diferentes clientes. A tabela 2.2 mostra a comparação de resultados entre a *Cloud* e a *Fog*.

Tabela 2.2 - Comparação de resultados entre a Cloud e a Fog

Aspectos Investigados	Cloud Tradicional	Fog Computing
Previsão de latência	5 segundos	1,5 segundos
Latência de exibição de página da WEB	8 segundo	3 segundos
Tráfego de internet	75 Kbps	10 Kbps
Hardware usado	Amazon WEB Server	Sistemas embarcados, a exemplo do Raspberry PI

Fonte: AL-DOGHMAN (2016)

Nesta tabela 2.2 observa-se em que todos os aspectos analisados tiveram melhores resultados no uso da *Fog Computing* em relação a *Cloud Computing*, isso demonstra sua eficiência entre a comunicação da *Cloud* e à IoT.

## 2.5 Plataformas para Sistemas Embarcados

Um sistema embarcado é um sistema microprocessado no qual o computador é completamente encapsulado ou dedicado ao dispositivo ou sistema que ele controla. Diferentemente de computadores de propósito geral, como o computador pessoal, um sistema embarcado realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos. Já que o sistema é dedicado a tarefas específicas, através de engenharia pode-se otimizar o projeto reduzindo tamanho, recursos computacionais e custo do produto.

Em geral tais sistemas não podem ter sua funcionalidade alterada durante o uso. Caso queira-se modificar o propósito é necessário reprogramar todo o sistema. A tabela 2.3 apresenta as características de quatro das principais plataformas de dispositivos para sistemas embarcados e seus respectivos modelos.

Tabela 2.3 - Comparação entre as plataformas embarcadas

PLACA	CPU	GPU	MEMÓRIA	COMUNICAÇÃO	SO	CUSTO
<b>Raspberry PI Zero</b>	1 Core 1GHz	250Mhz	512MB DDR2	1 USB	Linux	US\$ 5
<b>Raspberry PI 2 B</b>	4 Cores 0.9GHz	250Mhz	1GB DDR2	4 USB Ethernet	Linux	US\$ 35
<b>Raspberry PI 3 B</b>	4 Cores 1.2GHz	400Mhz	1GB DDR2	4 USB / Wireless Bluetooth / Ethernet	Linux Windows	US\$ 35
<b>Banana PI M2</b>	4 Cores 1GHz	355Mhz	1GB DDR3	5 USB / Wireless Ethernet	Linux Android	US\$ 46
<b>Banana PI M3</b>	8 Cores 1.8GHz	700Mhz	2GB DDR3	3 USB / Wireless Bluetooth / Ethernet	Linux Android	US\$ 74
<b>Orange PI One</b>	4 Cores 1.5GHz	600Mhz	512MB DDR3	2 USB Ethernet	Linux Android	US\$ 10
<b>Orange PI PC</b>	4 Cores 1.5GHz	600Mhz	1GB DDR3	4 USB Ethernet	Linux Android	US\$ 15
<b>Orange PI Plus</b>	4 Cores 1.5GHz	600Mhz	1GB DDR3	5 USB / Wireless Ethernet	Linux Android	US\$ 39
<b>Orange PI Plus 2</b>	4 Cores 1.5GHz	600Mhz	2GB DDR3	5 USB / Wireless Ethernet	Linux Android	US\$ 45
<b>Odroid C1+</b>	4 Cores 1.5GHz	600Mhz	1GB DDR3	5 USB Ethernet	Android	US\$ 32
<b>Odroid C2</b>	4 Cores 2GHz	700Mhz	2GB DDR3	5 USB Ethernet	Android	US\$ 40
<b>Odroid XU4</b>	8 Cores 2.1GHz	600Mhz	2GB DDR3	3 USB Ethernet	Linux Android	US\$ 74

Fonte: <http://www.sistemasembarcados.org> (adaptado)

Existem disponíveis no mercado muitas plataformas para sistemas embarcados com diferentes características, como poder de processamento, capacidade de memória e meios de comunicação, porém vamos analisar quatro dessas principais plataformas e os seus diferentes modelos, são elas: Raspberry Pi, Banana Pi, Orange Pi e Odroid.

Para selecionar a plataforma de dispositivo embarcado a ser utilizada em nosso trabalho, definimos cinco critérios para a seleção, que pudesse atender o objetivo final desta dissertação de mestrado que é, propor a implementação e análise de uma *Fog Computing*, para fornecer

StaaS (*Storage as a Service*), a dispositivos IoT utilizando plataforma de sistema embarcado, são eles:

1. **PROCESSAMENTO** – Possuir os requisitos mínimos para permitir a instalação das plataformas de nuvem selecionada para fornecer o serviço StaaS (*Storage as a Service*), que no caso foi a OwnCloud;
2. **MEMÓRIA** – Ter no mínimo 1GB de memória RAM, já que a plataforma de nuvem OwnCloud necessita de 512MB para sua instalação;
3. **COMUNICAÇÃO** – Possuir o máximo de conexões disponíveis, principalmente por WSN (rede de sensores sem fio), pelo fato de que uma *Fog Computing* poderá se comunicar com diversos dispositivos IoT heterogêneos;
4. **SISTEMA OPERACIONAL** – Deverá ter sido homologado o uso do sistema operacional Linux pelo fabricante, já que a plataforma OwnCloud *server* só funcionam com esse determinado sistema operacional;
5. **CUSTO** - Possuir um bom custo benefício, permitir oferecer o serviço StaaS (*Storage as a Service*) de modo satisfatório, com um custo baixo ao final do projeto.

Conforme mostrado na tabela 2.3 no primeiro critério todas as plataformas atendem a esse requisito, no segundo critério duas plataformas não atendem o requisito, a Raspberry Pi Zero e Orange Pi One, já no terceiro critério só duas plataformas atendem esse requisito, Raspberry Pi B e a Banana Pi M3, com mais possibilidades de comunicação USB, Wireless, Bluetooth e Ethernet, no quarto critério só duas plataformas não atenderam a esse requisito, a Odroid C1+ e a Odroid C2, no quinto critério todas as plataformas atenderam a esse requisito, por custarem um valor relativamente baixo. Sendo assim, escolhemos para utilização nesta dissertação de mestrado as plataformas de sistemas embarcados Raspberry PI modelo B e Banana PI modelo M3, por atenderem todos os critérios pré-definidos anteriormente.



## 2.6 Considerações Finais do Capítulo

Com o objetivo de proporcionar uma análise da contextualização teórica, foi feito um estudo em alguns *reviews* relacionados ao tema dessa dissertação. Esses temas foram uma análise sobre a *Cloud Computing* e seus serviços, as plataformas da computação em nuvem, uma comparação entre elas, foram abordados o conceito e a necessidade da *Fog Computing*, apresentamos uma comparação entre as plataformas de dispositivos para sistemas embarcados.

Observamos que a IoT (Internet das Coisas) necessita de muitos serviços para ser uma realidade, por ter recurso de processamento e armazenamento muito restrito, a sua interconexão com a *Cloud Computing* é um fator relevante e de elevada atenção para pesquisadores do mundo inteiro. As questões de latência, big data, armazenamento e poder de processamento próximo dos dispositivos de borda, fez com que surgisse a *Fog Computing*. A *Fog Computing* é a interligação entre os dispositivos IoT e a *Cloud Computing*, e pode levar poder de processamento e armazenamento aos dispositivos IoT, sendo possível o surgimento de novas tecnologias e aplicações para poder reduzir essas deficiências encontradas hoje.

No quadro 2.1 vimos que as plataformas OwnCloud pode ser usada para fornecer StaaS (*Storage as a Service*), podendo ser uma boa alternativa para diminuir o problema de armazenamento de dados, que atinge hoje os dispositivos IoT, servindo como *Fog Computing* para armazenar dados desses dispositivos e enviar à *Cloud Computing*, com segurança e eficiência no transporte dos dados entre as redes. Utilizando sistemas embarcados em plataforma de dispositivo de baixo custo, em vez de usar potentes e caros servidores para exercer essa função.

### TRABALHOS CORRELATOS

Nos últimos anos, a Internet mudou dramaticamente, tornou-se muito mais poderosa, a computação em nuvem é uma das causas. A computação em nuvem utiliza a tecnologia de virtualização, os provedores permitem que os usuários aluguem recursos de computação com um modelo "pague e use" para executar suas tarefas na nuvem.

A computação em nuvem tem sido objeto de muitas pesquisas se tornando muito popular nos últimos anos. Por outro lado, cada vez mais dispositivos são capazes de se conectar à Internet, e esses dispositivos conectados não só podem solicitar recursos da Internet, mas também enviar os dados para um destino desejado.

O termo "Internet das Coisas" (IoT) também é recente e hoje é uma área de pesquisa muito popular. O relatório de pesquisa mostra que mais de 30 bilhões de dispositivos IoT (Internet das Coisas) se conectarão através da rede sem fio (FAN, 2016). Mesmo que os dispositivos tenham capacidade de computação limitada, a computação em nuvem tem poder de fornecer computação em massa que pode ser utilizado facilmente.

Os dispositivos de IoT se tornaram mais poderosos, enquanto mais e mais dispositivos facilmente podem acessar a Internet, que se torna congestionada, por exemplo, um dispositivo tem de enviar a sua solicitação para uma nuvem dedicada e aguardar a sua resposta para efetuar a ação seguinte, podendo ocasionar uma latência muito alta.

A *Fog Computing* pode ser a chave para a solução. A *Fog Computing* utiliza os dispositivos de borda e oferece mais possibilidades de conexão e processamento, com dispositivos de rede que fornece o ponto de entrada para as redes principais. Os dispositivos de borda na rede *Fog Computing* são atualizados adicionando assim mais capacidade de computação e armazenamento à IoT.

Com a capacidade na borda da rede, algumas das funções do servidor podem ser descentralizadas para esses dispositivos, por exemplo, um dispositivo de borda pode coletar

todos os dados de sensores dentro de uma rede e enviá-los periodicamente como um lote, para reduzir a carga na largura de banda e a carga de processamento do servidor (MACHADO, 2017).

Nesse sentido, com a finalidade de sintetizar os estudos realizados na temática da implementação de uma *Fog Computing*, para fornecer serviço de armazenamento a dispositivos IoT, utilizando plataforma embarcada como Raspberry PI, e, ao mesmo tempo, verificar quais métodos ou técnicas estão presentes nessas metodologias, neste capítulo apresenta-se uma Revisão Sistemática, o estudo e análise sobre a literatura existente deste novo paradigma que é a *Fog Computing*.

### 3.1 Revisão Sistemática

Para alcançar um grau de rigor científico, procurou-se assegurar o processo de investigação a partir das concepções sobre Revisão Sistemática (RS) conforme descreve-se a seguir.

A revisão sistemática de literatura, identifica, avalia e interpreta todas as pesquisas disponíveis relevantes para uma questão específica, área temática, ou fenômeno de interesse (KITCHENHAM, 2004).

A importância do estabelecimento de um processo de revisão sistemática se concretiza ao observar a definição de fases, quais sejam: a) Planejamento da revisão; b) Procedimentos de condução e extração da RS e, ao final, c) Procedimentos de elaboração de relatórios da RS.

Dessa forma, os resultados são mais confiáveis em relação à revisão de literatura primária, em virtude de sua forma rigorosa e que dá possibilidade de repetições e auditoria.

Assim, a RS teve o seu foco na caracterização dos processos de desenvolvimento de solução para implementação de *Fog Computing*, que utilizassem plataforma embarcada como o Raspberry PI, e que fizessem uso de *Benchmark* para avaliar o serviço disponibilizado.

Durante a fase de planejamento da RS define-se primeiramente um protocolo, no qual constam o problema de pesquisa, de forma a delimitar a natureza essencial e a eficiência de determinadas práticas ou métodos no domínio em pauta, bem como aspectos como fontes de pesquisa, *Strings* de busca utilizadas (sequência ou combinação de palavras-chaves para a pesquisa), critérios de inclusão e exclusão de trabalhos encontrados na literatura e forma de sumarização dos resultados.

Na fase de condução da RS executa-se a formalização de um processo de pesquisa conforme o protocolo estabelecido. Nessa etapa, de forma sumarizada, devem ser listados todos os trabalhos encontrados na literatura e aplicados os critérios de inclusão e exclusão definidos no protocolo. O resultado final é uma lista de artigos que devem ser incluídos na fase seguinte.

Inicia-se então a fase de extração que consiste em um detalhamento das informações por meio da extração de variáveis e anotações específicas de referências que podem agregar valor ou alterar os rumos do processo de pesquisa desenvolvido até então. Ao final, desenvolve-se um relatório incluindo-se todas as etapas do processo de revisão sistemática, conclusões e resultados encontrados.

### 3.2 Planejamento da Revisão Sistemática

Para planejamento da revisão formulou-se inicialmente um protocolo de RS, no qual se delineou os objetivos, questões da pesquisa e fontes de buscas no Portal de Periódicos da CAPES (ACM, IEEE, Science Direct, Springer e SCOPUS), além de referências acadêmicas presentes no *Google Scholar* para iniciar a atividade de construção do protocolo de revisão.

Como questão de pesquisa, definiu-se para o processo investigatório em questão um estudo sobre duas dimensões: a) Questão primária: Qual plataforma de nuvem está sendo utilizada para implementar a *Fog Computing*? e b) Questão secundária: Qual plataforma de nuvem é suportada para implementação numa plataforma embarcada como Raspberry PI?

Para responder essas duas dimensões, consolidaram-se termos de buscas formuladas em inglês, em dois segmentos de pesquisa com a definição das seguintes palavras-chaves: Pesquisa I ("*Fog Computing Implementation*") e Pesquisa II ("*Cloud and IoT Integration*" ou "*Implementation of Cloud Computing in Raspberry PI*").

A justificativa de segmentos de palavras-chave em dois procedimentos de recuperação de informação é feita considerando-se a necessidade de avaliar as informações presentes no campo das especificações da implementação da *Fog Computing*, e de forma mais abrangente não excluindo as contribuições de outras tecnologias de comunicação de dados.

Nesse sentido, foram delineados dois procedimentos de pesquisa e, após avaliação interseccionou-se as fontes encontradas da pesquisa I e II e na qual os trabalhos recuperados em redundância foram preservados e isolados para fins estatísticos.

Para a execução dos procedimentos de busca, definiram *strings* para as fontes de busca escolhidas, conforme apresenta-se a seguir:

SCOPUS:

TITLE-ABS-KEY (("Fog Computing" OR "Fog Computing Implementation" OR "Cloud and IoT Integration" OR "Cloud Computing Platform Comparison" OR "Implementation of Cloud Computing in Raspberry PI"))

Ano de publicação: 2010 – 2017

IEEE

TITLE-ABS-KEY (("Fog Computing" OR "Fog Computing Implementation" OR "Cloud and IoT Integration" OR "Cloud Computing Platform Comparison" OR "Implementation of Cloud Computing in Raspberry PI"))

Ano de publicação: 2010 – 2017

ACM

TITLE-ABS-KEY (("Fog Computing" OR "Fog Computing Implementation" OR "Cloud and IoT Integration" OR "Cloud Computing Platform Comparison" OR "Implementation of Cloud Computing in Raspberry PI"))

Ano de publicação: 2010 – 2017

SPRINGER

TITLE-ABS-KEY (("Fog Computing" OR "Fog Computing Implementation" OR "Cloud and IoT Integration" OR "Cloud Computing Platform Comparison" OR "Implementation of Cloud Computing in Raspberry PI"))

Ano de publicação: 2010 – 2017

SCIENCE DIRECT

TITLE-ABS-KEY (("Fog Computing" OR "Fog Computing Implementation" OR "Cloud and IoT Integration" OR "Cloud Computing Platform Comparison" OR "Implementation of Cloud Computing in Raspberry PI"))

Ano de publicação: 2010 – 2017

GOOGLE SCHOLAR

TITLE-ABS-KEY (("Fog Computing" OR "Implementação da Fog Computing" OR "Integração da Cloud and IoT" OR "Comparação das Plataformas Cloud Computing" OR "Implementação de uma Nuvem de Armazenamento Privada"))

Ano de publicação: 2010 – 2017 – Apenas na linguagem português

A fim de estabelecer os interesses da pesquisa, o protocolo construído definiu os seguintes critérios de seleção de trabalhos: artigos publicados no período de 2010 a 2017; artigos completos disponíveis *online*; idioma inglês e português; estudos que abordem áreas da *Fog Computing*; e estudos que abordem a integração da *Cloud* à IoT; estudos que abordem a comparação entre plataformas *Cloud Computing*; e a implementação da *Cloud Computing* em plataforma embarcada, como o Raspberry PI.

De maneira a observar a relevância dos trabalhos selecionados, foram elaborados critérios de inclusão e exclusão, conforme segue:

Critérios de inclusão: trabalhos que abordem a questão relacionada à comunicação de redes com e sem fio; trabalhos que abordem a questão relacionada à *Fog Computing*; trabalhos que abordem a integração da *Cloud* à IoT; trabalhos que abordem a implementação da *Cloud Computing* em plataforma embarcada; e, trabalhos publicados em conferências e revistas especializadas.

Critérios de exclusão: trabalhos que não apresentem texto completo; trabalhos onde os termos de busca não se apresentem nos campos: título, resumo e palavras-chave; trabalhos de conclusão de curso; e, trabalhos que não abordem a questão relacionada ao critério de inclusão.

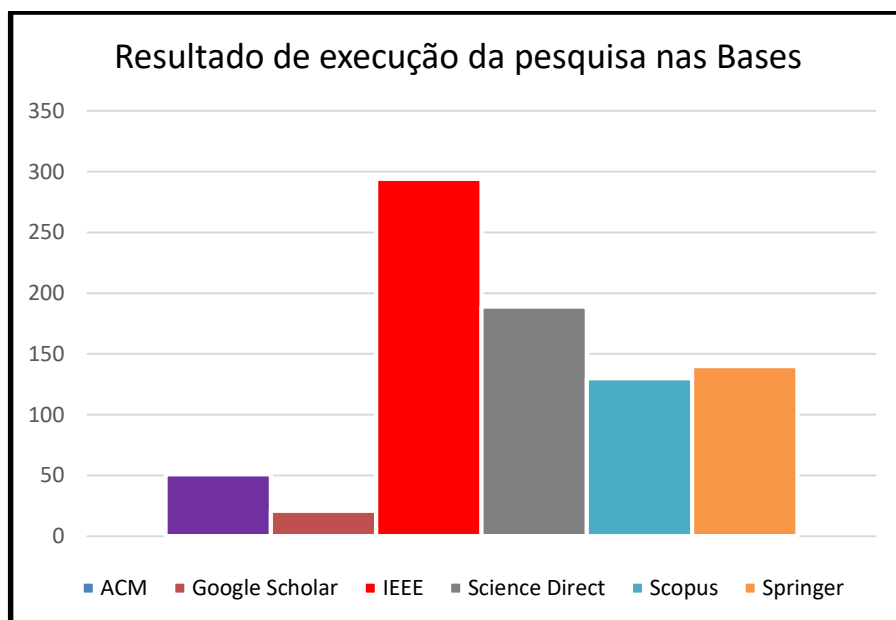
### 3.3 Execução da Pesquisa

Após a etapa de planejamento, o processo de execução da revisão sistemática (RS) foi iniciado. Na execução da RS os trabalhos foram recuperados por meio das *strings* já delineadas no protocolo de RS. Os trabalhos foram importados a partir de cada base de pesquisa em formato apropriado (de modo geral, no formato PDF) e armazenados, contendo as informações sobre o trabalho como: autor, título, palavras-chaves, resumo, ano de publicação e referências bibliográficas.

As *strings* de busca elaboradas no processo de planejamento foram executadas e utilizadas em cada um dos engenhos de busca das fontes selecionadas para um formato de importação compatível.

Foram identificados 825 artigos (gráfico 3.1), dos quais, em cada base totalizaram: ACM (51 artigos), GOOGLE SCHOLAR (21 artigos), IEEE (294 artigos), SCIENCE DIRECT (189 artigos), SCOPUS (130 artigos) e SPRINGER (140 artigos).

Gráfico 3.1 - Artigos identificados (fase de execução)



Fonte: Própria do autor (2017)

Os artigos foram identificados em um repositório de informações eletrônico para sua avaliação na segunda fase da RS, conforme se descreve a seguir. Observou-se que a grande quantidade de artigos relacionados na base IEEE (294 artigos) se deu em virtude de seu engenho de busca tratar todos os campos que forma a *string*, como palavras para pesquisa em todo o conteúdo dos artigos, o que dificultou o processo inicial de seleção destes artigos na revisão sistemática.

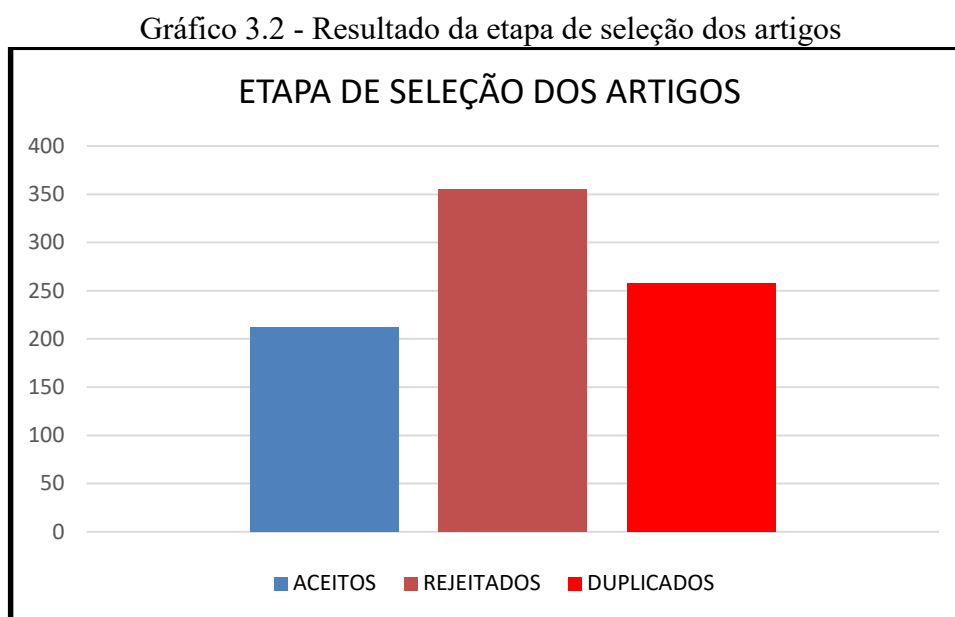
### 3.4 Procedimentos de Seleção e Extração

Uma vez que o protocolo de RS foi definido e os artigos foram identificados e armazenados na etapa de condução passa a ser executada, iniciando-se por uma seleção de informações e encerrando-se com a submissão dos trabalhos para a próxima fase da RS.

Nessa etapa, utilizando-se de um formulário eletrônico os pesquisadores efetuaram a leitura do resumo de cada artigo disponibilizado. Nesse momento, houve o julgamento dos

critérios de inclusão e exclusão dos artigos por meio de um processo de seleção, conforme se observa no gráfico 3.2.

Dos 825 trabalhos relacionados no processo de identificação, 212 trabalhos foram aceitos para ser avaliados na etapa de extração da revisão de literatura, assim 613 foram excluídos, sendo 355 pelos critérios definidos no protocolo e 258 trabalhos excluídos (gráfico 3.2) por estar duplicados quando da exportação dos resultados de consultas das bases.



Fonte: Própria do autor (2017)

Na etapa de extração, foram coletadas informações pré-definidas no protocolo de RS como: resumo do trabalho, objetivo do trabalho, resumo do experimento realizado, local do experimento, participantes, material, descrição da metodologia de avaliação, descrição de técnicas ou métodos de avaliação, técnicas de ensino, escopo do trabalho, objetos centrais de avaliação, parâmetros ou métricas de avaliação utilizadas, hipóteses ou questão problema evidenciada, natureza do método ou técnica de avaliação, área de aplicação/domínio, classificação do ambiente, resultados, referências relevantes, observações e trabalhos futuros.

Os 212 trabalhos aceitos por, pelo menos, um critério de seleção e conduzidos à etapa de extração para o processo de leitura completa, onde conclui-se a condução de 30 trabalhos, os quais atenderam os critérios formalizados no protocolo da revisão sistemática. Os 182 trabalhos já excluídos não atenderam a todos os critérios da etapa de análise na extração.



### 3.5 Resultados da Revisão Sistemática

Obtidos os resultados da etapa de extração, a revisão sistemática permitiu discutir aspectos específicos de cada trabalho, envolvendo os objetivos, aspectos metodológicos, métodos ou técnicas, parâmetros, padrão de comunicação, agregação de dados para melhor organização da implementação do ambiente de aplicação em *Fog*.

Os dados são sumarizados e apresentados no quadro 3.1 e foram organizados na ordem crescente cronologicamente, para demonstrar a evolução em relação ao tema *Fog Computing*. No quadro 3.1, os artigos foram analisados quanto ao objetivo da pesquisa, implementação da *Fog Computing*, a plataforma de *Cloud* usada, utilização de dispositivo embarcado, a descrição de técnicas ou métodos de avaliação (*benchmark*), trabalho futuro e também a observação em relação a contribuição deste trabalho.

No quadro 3.1 é possível observar que a *Fog Computing* é um novo paradigma e, amplamente pesquisada atualmente, prova disso é que 33,33% dos artigos selecionados para a pesquisa são do ano de (2017), publicados até o início daquele ano, isso mostra a ascensão do assunto nos dias de hoje.

CHEN (2017), afirma que a *Fog Computing* distribuirá serviços avançados de computação, armazenamento, rede e gerenciamento mais próximos dos usuários finais ao longo da borda da rede, da *Cloud* para dispositivos IoT, formando assim uma plataforma distribuída e virtualizada.

Com isso surgem os desafios de pesquisas, como apresentado por BOTTA (2016), destaca os seguintes assuntos: A padronização, potência e eficiência energética, big data, segurança e privacidade, inteligência, metodologia de integração, comunicações de rede, armazenamento, escalabilidade e flexibilidade.

Percebe-se que poucos artigos fizeram a implementação da *Fog Computing*, apenas 20% dos trabalhos. Nenhum deles disponibilizou o serviço StaaS (*Storage as a Service*). Os restantes fizeram pesquisas teóricas sobre o tema ou implementaram somente em nível de *Cloud Computing*, sem observar o conceito da *Fog* e a maioria não utilizou técnica ou método de avaliação como *benchmark*.

A análise dos estudos apresentados no quadro 3.1 permitiu identificar que a produção científica sobre a implementação da *Fog Computing* é escassa, esse percentual ainda é menor quando se trata de implementação com dispositivos de sistemas embarcados, tendo em vista que somente 13,33% dos estudos faz referência a esse tipo de implementação.

Dos artigos analisados podemos destacar dois, que este trabalho será uma extensão destas pesquisas, ANTONI (2015) e PRINCY (2015), apresentaram uma solução de implementação de nuvem privada de armazenamento, ao implementar o Owncloud utilizando Raspberry PI, sendo que não utilizaram técnica ou método de avaliação como *benchmark* específico para o serviço disponibilizado StaaS (*Storage as a Service*), nem foi aplicado o conceito de *Fog Computing*. Como apresentado por CRACIUNESCU (2015), a *Fog Computing* concentra-se na descarga de algumas das tarefas (por exemplo, tarefas de armazenamento ou processamento) geralmente realizadas na nuvem até a borda da rede, entre a *Cloud* e IoT.

Desse modo, nota-se a necessidade de maiores investimentos em pesquisa nesta temática, dada a importância da implementação da *Fog Computing* como tecnologia que promoverá mais segurança e velocidade na transmissão dos dados entre a *Cloud* e IoT.

Quadro 3.1 – Sumários dos artigos analisados

(conclusão)

Artigos Autor / Ano	Objetivo do trabalho	Implementação FOG	Plataforma Cloud	Dispositivo embarcado	Benchmark	Observação	Desafio de pesquisa ou Trabalho futuro
ZHANG (2010)	Apresentar um levantamento da computação em nuvem, destacando seus conceitos, princípios arquitetônicos, implementação de ponta, desafios de pesquisa e identificar direções de pesquisas nesta área.						Pesquisar: Migração de máquina virtual; Gerenciamento de energia; Segurança de dados e tecnologias de armazenamento; e Novas arquiteturas de nuvens.
MARTINI (2013)	Apresentar um estudo de análise forense usando um aplicativo de nuvem de código aberto amplamente utilizado, a aplicação <i>storage as a service</i> (SaaS) OwnCloud, como um estudo de caso.		✓		✓	Fez análise com <i>Cloud forensics framework</i> e utilizou o ownCloud.	Observar outros produtos <i>Cloud SaaS</i> disponíveis, incluindo os hospedados no ambiente de nuvem pública.
ZHU (2013)	Apresentar a otimização da WEB dentro do contexto <i>Fog Computing</i> . Aplicou métodos existentes para otimização da WEB de uma maneira inovadora, de tal forma que esses métodos podem ser combinados com conhecimento exclusivo que está disponível apenas nos nós de borda ( <i>Fog</i> ).	✓					Aplicar seus conceitos propostos para desenvolver um sistema de prova de conceito.

HAJIBABA (2014)	Apresentar o advento de novas formas de computação distribuída, ele fornece uma definição para computação em <i>Cloud</i> , <i>Jungle</i> e <i>Fog</i> , e são determinadas as principais características, além disso, são ilustradas suas arquiteturas.						Pesquisar novos paradigmas de computação distribuída.
NWOBODO (2015)	Analisar algumas plataformas que são usadas para projetar arquitetura de computadores, que satisfaça os requisitos de uso modular a computação em nuvem, é feita uma comparação de quatro plataformas de nuvem, as plataformas: AbiCloud, Eucalyptus, Nimbus e OpenNebula <i>cloud</i> .		✓		✓	Fez uma análise dos resultados e analisou as plataformas AbiCloud, Eucalyptus, Nimbus e OpenNebula <i>cloud</i> .	Não mencionou.
BABU (2015)	Apresentar a necessidade de integração da nuvem e da Internet das coisas, um paradigma orientado por agente e assistido pela <i>Cloud</i> , no <i>Cloud IoT</i> , é proposta a arquitetura de referência para visão orientada a agente e a assistência a <i>Cloud</i> , identificar e discutir sobre questões abertas e direções futuras.						Pesquisar as questões que envolvem a CloudIoT como: Padronização; Complexidade da mineração de dados; Capacidade da Cloud e a <i>Fog Computing</i> .
STOJMENOVIC (2015)	Apresentar a motivação e as vantagens da <i>Fog Computing</i> e analisar suas aplicações em uma série de cenários reais, como o <i>Smart Grid</i> , semáforos inteligentes em redes de veículos e redes definidas por <i>software</i> , discutir o estado da arte da <i>Fog Computing</i> , as questões de segurança e privacidade de acordo com o atual paradigma da <i>Fog Computing</i> .						Pesquisar <i>Fog Computing</i> em <i>Smart Grid</i> ; SDN em redes de veículos; Mobilidade entre <i>Fog</i> e <i>Cloud</i> e a Segurança.
HUO (2015)	Apresentar o design e implementação de uma plataforma de armazenamento em nuvem privada utilizando OpenStack. Projetada com base na estrutura Python Django e no <i>framework</i> Swift, adotada para construir o sistema do site para completar as tarefas de criação, <i>upload</i> e <i>download</i> de arquivos de dados corporativos, baseado na WEB por meio de método para prover a viabilidade do sistema.		✓			Utilizou a plataforma OpenStack.	Não mencionou.

ANTONI (2015)	Apresentar implementação de uma nuvem de armazenamento privada usando Owncloud e Raspberry PI, e destacar a importância e vantagens de se utilizar uma nuvem de armazenamento privada.		✓	✓	✓	Fez uma análise com a ferramenta Cacti, utilizou a plataforma Owncloud e utilizou dispositivo embarcado Raspberry PI.	Não mencionou.
PRINCY (2015)	Apresentar implementação de um servidor de nuvem privada configurado em um Raspberry Pi, que pode ser usado como um dispositivo de armazenamento para aplicações envolvendo sinais em tempo real, converter e transmitir o sinal analógico para digital, em série do Arduino para o Raspberry Pi.		✓	✓		Utilizou a plataforma Owncloud e utilizou dispositivo embarcado Raspberry PI.	Não mencionou.
CRACIUNESCU (2015)	Apresentar uma implementação em laboratório de e-Saúde, onde o processamento em tempo real é realizado pelo PC doméstico, enquanto os metadados extraídos são enviados para a nuvem para processamento posterior.	✓	✓	✓		Utilizou dispositivo embarcado Arduino como sensor.	O trabalho futuro incluirá a adição de mais sensores e adição de dispositivos off-the-shelf, que são atualmente <i>mainstream</i> , e os dados de fusão desses dispositivos.
LONGO (2015)	Descreve a abordagem e as soluções até o momento implementadas preliminarmente para permitir interações mediadas por nuvem com unidades de nós <i>hosts</i> , para hospedar sensores e atuadores, propondo Stack4Things, uma estrutura para Sensing-and-Actuation-as-a-Service.		✓			Utilizou plataforma Cloud OpenStack.	O trabalho futuro abordará as comunicações seguras ao atualizar as comunicações baseadas no WebSocket para o Secure WebSockets (was).
BOTTA (2016)	Fornecer uma pesquisa da literatura sobre a integração da <i>Cloud</i> e da IoT, começando por analisar os conceitos básicos de IoT e <i>Cloud Computing</i> , discutindo a sua complementaridade, detalhando o que está atualmente disponível para gerar sua integração.						Desafios: Segurança e privacidade; Heterogeneidade; Performance; Confiabilidade; Aspectos jurídicos e sociais; <i>Big data</i> , Redes de sensores, Monitoramento e Fog computing.

AL-DOGHMAN (2016)	Apresenta conceito, interesse, abordagens e práticas da <i>Fog Computing</i> e descrever a necessidade de adotar esse novo modelo e investigar seus principais recursos ao elucidar os cenários para implementá-la, descrevendo assim sua importância no mundo IoT.						Não mencionou.
STEINER (2016)	Propor a <i>Fog Computing</i> como um meio arquitetônico para realizar IIoT e discutir duas tecnologias habilitadas para <i>Fog Computing</i> : virtualização e comunicação determinística.						Pesquisar mais o tema <i>Fog Computing</i> .
KUMAR (2016)	Aborda a ameaça de problemas de segurança, especialmente com privacidade de localização e confidencialidade de dados, discutir a forma como os provedores de serviços, bem como o governo podem acessar os dados dos usuários e referenciar o conceito de técnica de armadilha com algumas modificações para localização e privacidade de dados.						Projetar de uma maneira que só precisamos criar um nó de <i>Fog</i> falso e, de acordo com o tráfego, esse nó falso se replica automaticamente, dificultando a identificação da localização.
JAIN (2016)	Apresentar um <i>overview</i> da <i>Fog Computing</i> , define o que é a <i>edge computing</i> e como pode ser alcançada através da <i>Fog Computing</i> .						Pesquisar os aspectos da <i>Fog Computing</i> em: <i>Smart Grid</i> e SDN em redes veiculares.
VERBA (2016)	Analisar as plataformas existentes e suas deficiências, bem como propor uma plataforma de <i>gateway</i> modular, baseada em mensagens que permite o agrupamento de <i>gateways</i> e a abstração dos detalhes do protocolo de comunicação periférica.	✓	✓	✓	✓	Avaliou com IBM <i>benchmarck</i> , utilizou plataforma <i>Cloud Openstack</i> e utilizou dispositivo embarcado Raspberry PI.	Consistirá em desenvolver um ambiente de laboratório inteligente com diversos dispositivos mais complexos e cenários de controle para testar completamente o sistema.

DÍAZ (2016)	Apresentar uma pesquisa de componentes de integração: plataformas em nuvem, infraestruturas em nuvem e <i>middleware</i> IoT, além disso, algumas propostas de integração e técnicas de análise de dados, bem como diferentes desafios e questões de pesquisas.						Pesquisar questões da integração <i>Cloud</i> a IoT como: Privacidade e segurança; Falta de interoperabilidade; <i>Fog Computing</i> e IPV6.
FAN (2016)	Apresentar a capacidade do recurso de WEB <i>Caching</i> adicionado ao dispositivo de borda para servir como servidor <i>proxy</i> de armazenamento em <i>cache</i> , para obter mais armazenamento em <i>cache</i> , os dispositivos finais também são explorados para fornecer algum espaço de cache.	✓			✓	Fez avaliação.	Adicionar mais funcionalidades ao dispositivo de borda, como a segurança. Implementar o trabalho no mundo real.
MRÓWCZYŃSKI (2017)	Apresentar uma estrutura de <i>benchmarking</i> e monitoramento para sincronização de arquivos e serviços de compartilhamento, para fornecer métricas de desempenho e robustez para sincronização de arquivos interconectados e serviços de compartilhamento.		✓		✓	Fez avaliação com <i>benchmark</i> Smashbox e utilizou as plataformas <i>Cloud</i> e OwnCloud e OpenStack.	Não mencionou.
HAO (2017)	Apresentar uma descrição detalhada da <i>Fog Computing</i> , explorar seus desafios e problemas de pesquisa, apresentar um <i>design</i> do WM-FOG, uma estrutura de computação para ambientes <i>Fog</i> , que engloba essa arquitetura de <i>software</i> e avalia seu protótipo do sistema.	✓	✓		✓	Fez avaliação e utilizou plataforma <i>Cloud</i> .	Trabalho futuro envolverá a adição de mais recursos ao WM-FOG para melhor atender às aplicações de <i>Fog Computing</i> .
CHIANG (2017)	Apresentar 10 questões e resposta do paradigma <i>Fog Computing</i> e sua comunicação de rede, questões como definição, comparação entre <i>Fog</i> , <i>Edge</i> e <i>Cloud Computing</i> , segurança, vantagens e desafios de pesquisas.						Heterogeneidade e restrição; Interface <i>cloud-fog</i> ; Armazenamento elástico e volátil; e a Segurança.

LI (2017)	Discutir dois conceitos de codificação recentemente propostos, códigos de mínima largura de banda e códigos mínima latência, e ilustram seus impactos na <i>Fog Computing</i> , também analisar uma estrutura de codificação unificada que inclui as duas técnicas de codificação acima descritas.					✓	Fez avaliação.	Computação heterogênea; Redes com topologia de camada múltipla e estruturada; Tarefas de computação em várias etapas; Custos de computação codificados; Verificar a computação distribuída; Explorar as estruturas algébricas das tarefas computacionais; Aplicações pesadas de comunicação e Nós de <i>fog</i> plug-and-play.
LINTHICUM (2017)	Apresentar uma análise da conexão da <i>Fog</i> e <i>Cloud</i> computing definindo problemas e soluções, referentes a sua padronização.							Padronizar a arquitetura <i>Fog Computing</i> .
MCMILLIN (2017)	Analisar a <i>Fog Computing</i> para o desenvolvimento e sustentabilidade da vida inteligente.							Pesquisar a Segurança e Privacidade em <i>Fog Computing</i> .
ALRAWAIS (2017)	Discuti os problemas de segurança e privacidade no ambiente IoT, propor um mecanismo que emprega a <i>Fog</i> para melhorar a distribuição de informações na revogação de certificados nos dispositivos IoT, aprimorando a segurança.	✓	✓			✓	Fez avaliação.	Privacidade; Atualização de dispositivos IoT; Protocolos seguros e eficientes; Autenticação; Detecção de ataque; Verificação de localização e Controle de acesso.
CHEN (2017)	Apresentar a <i>Fog Computing</i> , como novo paradigma, que distribuirá serviços avançados de computação, armazenamento, rede e gerenciamento mais próximos dos usuários finais ao longo da borda da rede, da <i>cloud</i> para IoT, formando assim uma plataforma distribuída e virtualizada.							Rede Veiculares; Veículo com piloto automático; Cidades inteligentes; Redes industriais, Sistemas de petróleo e gás; e Sistemas de saúde.



OSANAIYE (2017)	Descrever a arquitetura de <i>Fog Computing</i> e revisar seus diferentes serviços e aplicativos, discutir questões de segurança e privacidade, apresentar uma abordagem conceitual de migração em tempo real de pré cópia para a migração de VM.						Implantação do <i>framework</i> no mundo real ou ambiente de teste, com o objetivo de sua validação.
POPENTIU- VLADICESCU (2017)	Analisar os modelos, arquiteturas e práticas existentes em <i>Fog computing</i> visando a confiabilidade e segurança dos sistemas de <i>Fog</i> , uma abordagem considerada integradora de três componentes da confiabilidade do sistema: a confiabilidade dos nós, a confiabilidade da rede e a confiabilidade do <i>software</i> , a arquitetura de referência OpenFog e os esquemas AJIA e BDSC.						Resolver problemas técnicos e de algorítmicos no paradigma de <i>Fog</i> .

Fonte: Própria do autor (2017)

### METODOLOGIA PARA PROTOTIPAÇÃO E COLETA DE DADOS

Neste capítulo são apresentados métodos utilizados para a prototipação e coleta dos dados obtidos para análise nos dispositivos implementados, tais como: metodologia utilizada, arquitetura de simulação ou cenário de testes, dispositivos de sistemas embarcados Raspberry Pi 3, Banana Pi M3 e o servidor DELL PowerEdge T410. Assim como a ferramenta de virtualização Docker *Container*, o sistema de *benchmark* Smashbox e o Zabbix, utilizado para coletar e analisar os dados de desempenho das implementações, as quais são detalhadas neste capítulo.

#### 4.1 Metodologia

Ao definir uma metodologia de avaliação de desempenho, deve-se ter atenção a fim de não cometer erros comuns, como inexistência de objetivos, propostas tendenciosas, incorretos métodos de avaliação, entre outros. Para evitar tais erros, o ideal é adotar uma abordagem sistemática como a proposta por Jain (1991), a qual foi aplicada neste trabalho. Para empregar essa metodologia torna-se necessário seguir uma sequência de passos.

O primeiro passo é a definição dos objetivos e do sistema. Pode ser difícil precisar os objetivos, todavia, é o primeiro passo para a resposta de um problema. O sistema deve ser claro, pois ele influencia diretamente nas métricas utilizadas para avaliação de desempenho, bem como no fluxo de carga para validação.

O segundo passo é a elaboração da lista de serviços e resultados esperados. Cada sistema provê um conjunto de serviços, e para cada um deles existe um conjunto de possíveis resultados.

Com o terceiro passo, tem-se a seleção de métricas que estabelecem os critérios para a comparação do desempenho. A fim de escolher boas métricas, Jain (1991) recomenda verificar

se as mesmas possuem baixa variação para diminuir o número de experimentos necessários; não redundância de modo a evitar números supérfluos; e completude com o intuito de retratar todos os resultados possíveis. Ainda segundo Jain (1991), as métricas mais comuns são: eficiência (capacidade utilizável sobre capacidade nominal), confiabilidade (tempo sem erros), disponibilidade (tempo médio entre falhas) e a relação custo/desempenho.

O quarto passo é a elaboração da lista de parâmetros que afetam o desempenho. Essa lista pode ser dividida em parâmetros de sistema (que geralmente não variam de uma instância para outra) e de carga (que são características das solicitações dos usuários).

Já o quinto passo, trata da escolha de fatores para estudo, que são parâmetros os quais sofrerão variações durante a pesquisa. Esses fatores podem assumir valores que são denominados níveis. Recomenda-se iniciar com poucos fatores e poucos níveis, e aumentar a lista conforme necessidade.

O sexto passo é a seleção da técnica de avaliação. Existem três técnicas: a simulação, a modelagem analítica e a medição. A técnica a utilizar dependerá do tempo e dos recursos disponíveis para a resolução do problema.

Com o sétimo passo, tem-se a escolha da carga, a qual consiste em uma lista de solicitações de serviços ao sistema. É importante que retrate o uso real.

Em seguida, o oitavo passo é o planejamento dos experimentos. Com a lista de fatores e níveis, deve-se estabelecer uma sequência de experimentos de modo a obter o máximo de informações possíveis com o mínimo de esforço realizado.

Com o nono passo, tem-se a análise e interpretação dos dados. Nessa etapa deve-se utilizar técnicas estatísticas adequadas a fim de consolidar os resultados obtidos, de modo a permitir realizar conclusões sobre o desempenho do sistema.

Já o décimo e último passo é a apresentação dos resultados. Nesta etapa deve-se atentar à apresentação final da avaliação.

#### **4.1.1 Aplicação da Metodologia**

Ao aplicar esta metodologia, nota-se sua importância diante da forma organizada na qual o trabalho foi conduzido. Inicialmente, torna-se necessário definir os objetivos, em seguida o escopo do sistema, os serviços oferecidos, e a técnica de avaliação. Como são mostrados a seguir.

**Objetivos:**

- Realizar uma análise de desempenho e eficiência em dois dispositivos de sistemas embarcados de baixo custo e um servidor de alto desempenho;
- Determinar fatores relevantes no que se refere ao desempenho destes equipamentos.

**Sistema:**

- O sistema corresponde a um *software* de nuvem para fornecer serviço de armazenamento StaaS (*Storage as a Service*) denominado OwnCloud, o mesmo interage com o meio através de transferência de arquivos entre o cliente e o servidor.

**Serviço:**

- Transferência de arquivo entre o sistema cliente e o sistema servidor.

**Técnica de avaliação:**

- Medição, pois trata-se de uma técnica útil para análise de desempenho de sistemas informáticos.

A atividade foi dividida em cinco etapas:

Etapas 1 - Projeto e cenário de testes;

Etapas 2 - Especificação das métricas;

Etapas 3 - Definição dos parâmetros, fatores e carga;

Etapas 4 - Planejamento e realização dos experimentos;

Etapas 5 - Análise estatística dos resultados obtidos.

A subseção 4.1.2 descreve as três primeiras etapas, enquanto a quarta e quinta são abordadas no próximo capítulo.

### 4.1.2 Projeto e Cenário de Testes

Para realizar a prototipação deve-se pressupor a existência de um modelo computacional idêntico ao ambiente real de produção. Na literatura, uma boa descrição para análise de desempenho em sistemas de nuvem para fornecer serviço de armazenamento StaaS (*Storage as a Service*) foi encontrada, como exemplo é possível citar o trabalho de MRÓWCZYŃSKI *et al.* (2017). Faz uma análise de desempenho para sincronização de arquivos, serviços de compartilhamento, e fornecer métricas de desempenho e robustez para sincronização de arquivos interconectados, utilizando o sistema de *benchmark* Smashbox.

Baseado nesse trabalho e em especificações técnicas de equipamentos disponíveis comercialmente, o presente trabalho foi realizado com dois dispositivos de sistemas embarcados modernos e um servidor com alto poder de processamento, e com a proposta de complementar as pesquisas já realizadas, e assim contribuir de forma efetiva para a indústria, para pequenas e grandes empresas. Bem como para a área acadêmica, pois amplia a realização de novas pesquisas.

Este trabalho verifica a possibilidade de atestar a utilização de dispositivos de sistemas embarcados modernos como servidores de um sistema de nuvem denominado OwnCloud para fornecer o serviço StaaS (*Storage as a Service*), faz a implementação e análise de uma *Fog Computing* para fornecer o serviço a dispositivos IoT. Podendo ser uma boa alternativa para sanar problemas relacionados ao armazenamento de dados dos dispositivos IoT, servindo como *Fog Computing* para armazenar dados desses dispositivos e enviar à *Cloud Computing*, com segurança e eficiência no transporte dos dados entre as redes, sendo comparando os resultados com de um servidor de alto desempenho.

A análise de desempenho consiste em verificar o tempo e velocidade de *upload* e *download* de cada teste realizado com diferentes quantidades e tamanhos de arquivos em cada equipamento analisado, e com diferentes combinações de sistemas operacionais. Também foi analisado o sistema não virtualizado e virtualizado com Docker *Container* para saber o seu impacto em apenas um nó. Para isso, utilizou-se a ferramenta de *benchmark* Smashbox para análise de desempenho para sincronização de arquivos, serviços de compartilhamento, utilizado por pesquisadores no meio acadêmico, como por exemplo, (MRÓWCZYŃSKI *et al.* 2017).

Foram realizados cinco tipos de diferentes testes para avaliar o desempenho dos equipamentos analisados, para o fornecimento do serviço StaaS, esses testes foram escolhidos por ser os mesmos utilizados por MRÓWCZYŃSKI (2017), exceto o Test4. O quadro 4.1 informa abreviaturas dos testes e sua correspondente distribuição de arquivos.

Quadro 4.1 - Abreviaturas dos testes

Nome	Abreviação	Quantidade de arquivos	Tamanho do arquivo	Volume total do teste
Test0	0/1/1	1	1 Byte	1 Byte
Test1	0/1/100000000	1	100 Megabytes	100 Megabytes
Test2	0/10/10000000	10	10 Megabytes	100 Megabytes
Test3	0/1000/10000	1000	10 Kilobytes	10 Megabytes
Test4	0/1/500000000	1	500 Megabytes	500 Megabytes

Fonte: Adaptado de MRÓWCZYŃSKI (2017)

Ainda em análise de desempenho, realizou-se o monitoramento do consumo de memória, processamento e do tráfego de rede em momentos de elevado uso do sistema; a fim de atingir o objetivo desejado utilizou-se o *software* de monitoramento Zabbix, o qual permitiu monitorar a eficiência dos equipamentos testados durante todo o experimento.

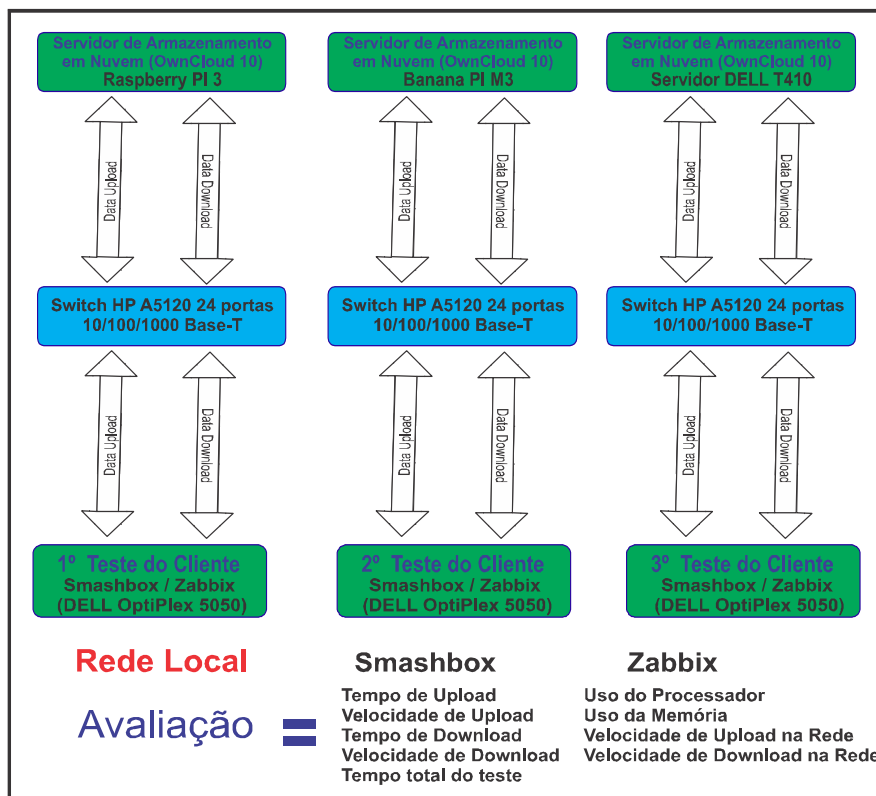
Para a execução do experimento foi desenvolvido um sistema denominado FogSys e um simulador para enviar as informações dos dispositivos IoT através de *Upload* e sincronizar com a *Cloud Storage Client* em *real time*, fazendo o papel de *Fog Computing* instalado no dispositivo de sistema embarcado Raspberry PI, e transferindo estes arquivos para a *Cloud Computing* que por sua vez sincroniza os dados com o dispositivo móvel, um *smartphone* utilizando um SO Android.

Para a realização deste trabalho, foi necessário elaborar um cenário de testes, de modo a possibilitar a execução de todos os objetivos propostos. Sendo assim, o cenário contemplou dois dispositivos de sistemas embarcados modernos e de baixo custo (Raspberry Pi 3 e Banana Pi M3), um Switch HP A5120 24 portas 10/100/1000 Base-T, um Desktop DELL OptiPlex 5050 Core i5 16GB RAM e 1TB de HD, um servidor DELL Power Edge T410 Xeon 16GB RAM e 2 HDs de 320GB, um *Acess Point* RUCKUS ZoneFlex 7363 e um Smartphone Samsung J7.

Na análise de desempenho o Switch HP A5120 24 portas 10/100/1000 Base-T, é responsável por interligar todos os equipamentos em rede. Cada dispositivo embarcado e o servidor DELL Power Edge T410 possui suporte ao sistema de nuvem OwnCloud para fornecer o serviço StaaS (*Storage as a Service*). O Desktop DELL OptiPlex 5050 foi alocado com o suporte ao *benchmark* Smashbox e a ferramenta de monitoramento de rede de computadores Zabbix, sendo esses sistemas virtualizados com Docker *Container* para ser o gerador de carga

e realizar a coleta dos dados. A figura 4.1 ilustra a arquitetura do cenário para a realização dos testes de avaliação.

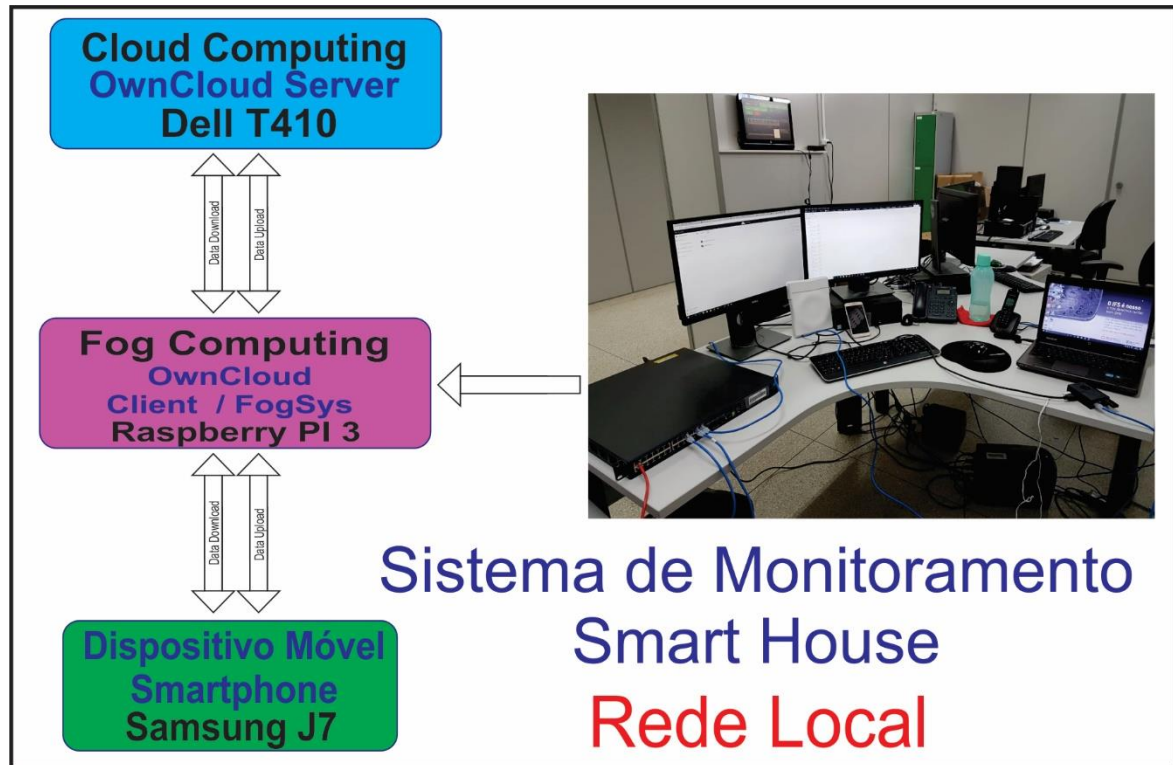
Fig. 4.1 - Cenário para a realização dos testes



Fonte: Própria do autor (2018)

No experimento foi simulado o monitoramento de ambiente *Smart House* com simulação de sensores IoT. Switch HP A5120 24 portas 10/100/1000 Base-T, é responsável por interligar todos os equipamentos em rede. O dispositivo de sistema embarcado Raspberry Pi com o suporte ao sistema de nuvem OwnCloud *Client* e o sistema FogSys instalados com virtualização, o servidor DELL Power Edge T410 possui suporte ao sistema de nuvem OwnCloud *Server* instalado para fornecer o serviço StaaS (*Storage as a Service*). O *Acess Point* RUCKUS ZoneFlex 7363 fornece acesso Wi-Fi para o *Smartphone* Samsung J7 com suporte ao sistema de nuvem OwnCloud *Client*. O dispositivo desktop DELL OptiPlex 5050 foi alocado com o suporte ao sistema FogSys com o módulo de simulador para dispositivos IoT, para ser o gerador de carga e simular um ambiente real, sendo que todos esses sistemas foram virtualizados com Docker *Container*, exceto o *Smartphone* Samsung Galaxy J7 que foi instalado o sistema de sincronização para o OwnCloud denominado cottonCloud versão 2.5.1. A figura 4.2 ilustra a arquitetura do cenário para a realização do experimento.

Fig. 4.2 - Cenário para a realização do experimento



Fonte: Própria do autor (2018)

Os dispositivos foram submetidos a momentos de elevado consumo do sistema, assim, foi possível atingir o objetivo deste trabalho com a realização das devidas simulações. Mais detalhes do sistema FogSys serão descritos no capítulo 6.

## 4.2 Raspberry PI Modelo B

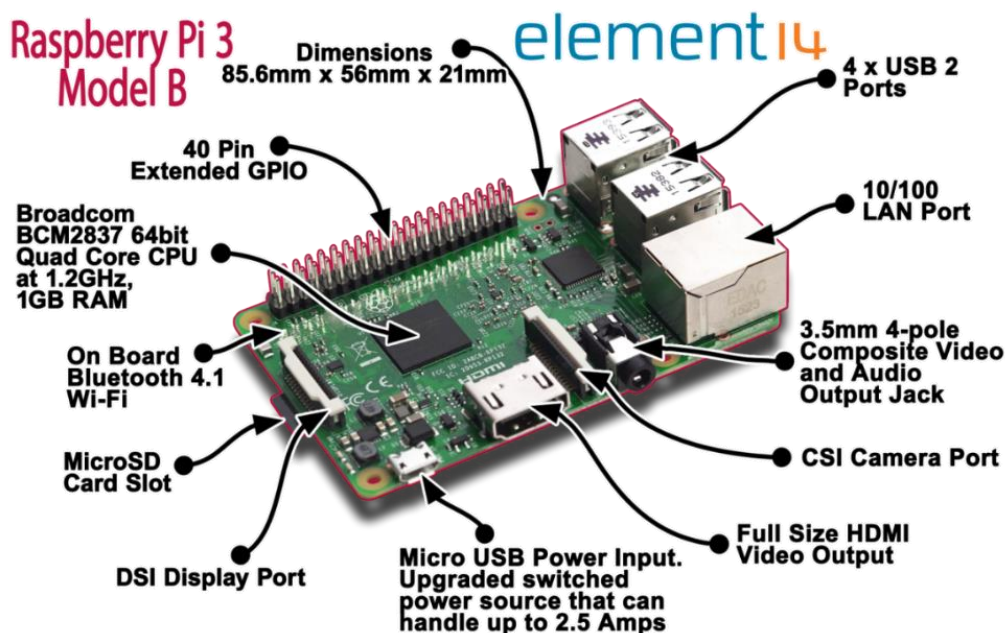
Raspberry PI é um pequeno computador potente baseado em microcontrolador ARM, ele funciona em distribuições Linux com o sistema operacional Raspbian distribuição Debian. Uma placa Raspberry PI suporta cartão de memória SD, teclado e mouse USB, monitor HDMI e fonte de alimentação, é possível fazer o Raspberry Pi funcionar como um computador normal de propósito geral (PRINCY, 2015).

Apesar do tamanho reduzido, medindo aproximadamente 9 x 6 x 2 cm ele vem equipado com 1GB de memória RAM, processador ARMv8 quadcore de 1.2 Ghz, armazenamento de 16GB (cartão micro SD) podendo ser expandido, suas expansões se dão através das quatro portas USB, uma porta Ethernet e uma HDMI (ANTONI, 2015).



O dispositivo Raspberry PI é similar a outros encontrados no mercado, a exemplo da Banana PI e Orange PI. A figura 4.3, demonstra mais características e detalhes do novo Raspberry PI 3 modelo B.

Fig. 4.3 - Raspberry PI 3 Modelo B



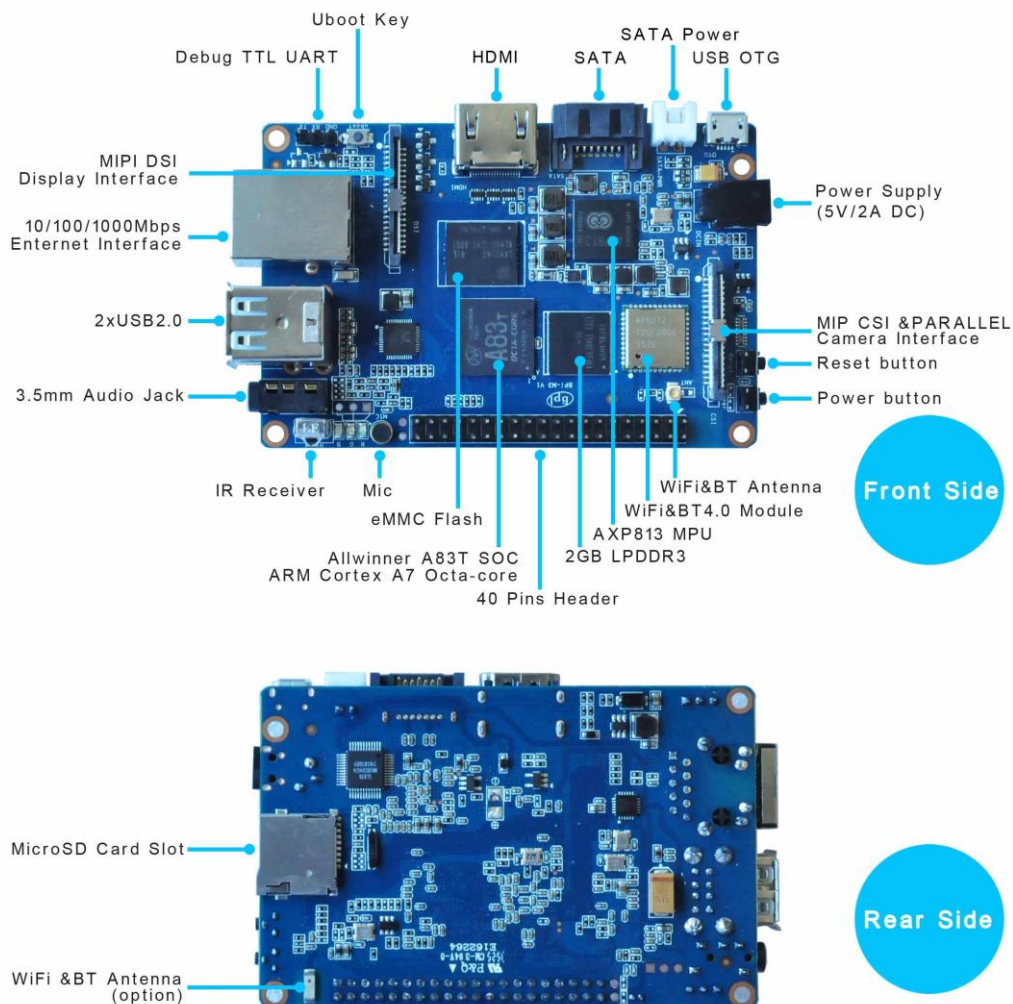
Fonte: <https://www.element14.com>

### 4.3 Banana PI Modelo M3

Banana PI M3 é um mine computador potente, com sua placa medindo 92mm x 60mm e um processador Octa-core e 2GB de RAM. Ao lado da unidade de processamento, possui uma conexão Gigabit Ethernet, 2 USB, SATA, WiFi, Bluetooth e HDMI. Ele pode ser executado em uma variedade de sistemas operacionais, incluindo Android, Lubuntu, Ubuntu, Debian e Raspbian.

Banana PI M3 é um dispositivo de plataforma aberta, é ideal para aplicações de jogos e para desenvolver novas tecnologias, em vez de simplesmente usar as tecnologias já existentes. Começar um projeto e construir servidores em mines computadores pode ser divertido e gratificante, podendo fazer diferente, inspirar os outros e construir projetos relevantes para a comunidade acadêmica e industrial. A figura 4.4, demonstra mais características e detalhes do Banana PI modelo M3.

Fig. 4.4 - Banana PI Modelo M3



Fonte: <http://www.banana-pi.org>

#### 4.4 Dell PowerEdge T410

O servidor Dell PowerEdge T410 conta com o desempenho dos processadores Intel Xeon séries 5500 e 5600, a disponibilidade de até 6 unidades de disco rígido (3,5" ou 2,5"), memória DDR3 e uma profundidade de chassi de 24" ideal para implantação e operação eficientes. O *design* do PowerEdge T410 é dotado de funcionalidade, incluindo tecnologia otimizada para reduzir o consumo de energia e gerenciamento simplificado de sistemas.

O PowerEdge T410 aproveita a excelente compatibilidade e confiabilidade de sistema da Dell. O *layout* consistente dos componentes e a disposição funcional das portas de interface e das fontes de alimentação facilitam a instalação e a reimplantação. Com uma profundidade de 24", o chassi do PowerEdge T410 é fácil de acessar e foi projetado para residir em uma instalação de *back-office*, varejo ou pequeno escritório, onde um chassi pequeno e uma acústica silenciosa são importantes.

Os resistentes suportes de metal das unidades de disco rígido e o cabeamento organizado foram projetados para melhorar o acesso aos componentes e o fluxo de ar em todo o servidor. O design funcional do PowerEdge T410 fornece uma acústica silenciosa e apresenta uma tela de LCD opcional posicionada na tampa frontal para facilitar o monitoramento. A figura 4.5, demonstra uma comparação visual da diferença de tamanho entre o servidor mais DELL PowerEdge T410, e os dispositivos de sistemas embarcados Banana PI M3 e o Raspberry Pi 3.

Fig. 4.5 - DELL T410, Banana Pi M3 e Raspberry Pi 3



Fonte: Própria do autor (2018)

## 4.5 Benchmark Smashbox

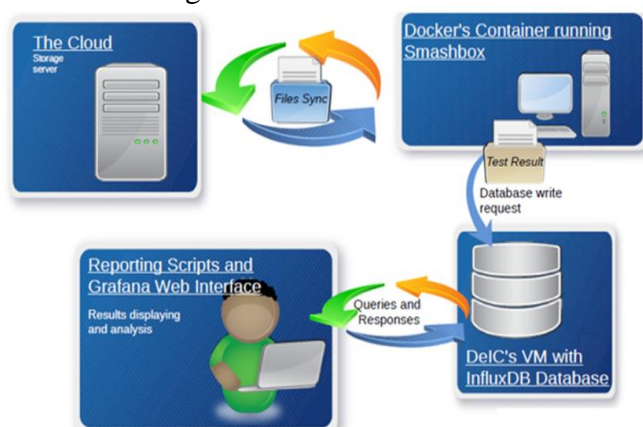
O Smashbox é uma estrutura de *benchmarking* e monitoramento para sincronização de arquivos e serviços de compartilhamento, permitindo aos provedores de serviços monitorar o status operacional de seus serviços, entendendo o comportamento do serviço sob diferentes tipos de carga e com diferentes locais de rede para a sincronização de clientes. A estrutura é

projetada como uma ferramenta de monitoramento e *benchmarking* para fornecer métricas de desempenho e robustez para a sincronização de arquivos interconectados e serviços de compartilhamento, como o *Open Cloud Mesh* (MRÓWCZYŃSKI, 2017).

O Smashbox foi inicialmente desenvolvido para testes funcionais de serviços baseados em OwnCloud, é uma estrutura de teste de ponta a ponta para funcionalidade principal de serviço de armazenamento, sincronização e compartilhamento de arquivos, pode ser executado de forma interativa a partir de uma linha de comando, através de acesso à área de trabalho ou via *scripts*. O conjunto padrão de testes do Smashbox inclui vários cenários de sincronização básica (*upload* / *download* / conflitos), operações simultâneas do cliente, condicionamento de transferência, consistência de compartilhamento, integridade do arquivo e concorrência de protocolo.

A estrutura original foi estendida com capacidades de monitoramento e relatórios para reportar informações de tempo para diferentes estágios do protocolo de sincronização OwnCloud, tempo total de execução dos testes e o número de erros e condições de falha. Ele possui um componente de monitoramento para capturar as taxas de transferência registradas na interface de rede local durante a sincronização, também oferece a possibilidade de conectar novos tipos de clientes de sincronização à estrutura (Dropbox e Seafile), essas adições estão integradas na estrutura Smashbox e estão disponíveis como recursos opcionais na configuração (MRÓWCZYŃSKI, 2017). A figura 4.6 demonstra o ambiente de monitoramento do *benchmarking* Smashbox.

Fig. 4.6 - Benchmarking e ambiente de monitoramento do Smashbox



Fonte: MRÓWCZYŃSKI (2017)

Atualmente, duas das soluções de sincronização de nuvem mais populares em redes de colaboração de pesquisa são OwnCloud e Seafile, a vantagem dessas soluções é que elas fornecem uma interface de usuário simples tipo Dropbox e também permitem um alto grau de personalização de *back-ends* e *front-ends*.

O monitoramento foi integrado ao serviço WEB do Grafana e ao InfluxDB, a visualização também é possível usando *scripts* python e matplotlib (MRÓWCZYŃSKI, 2017).

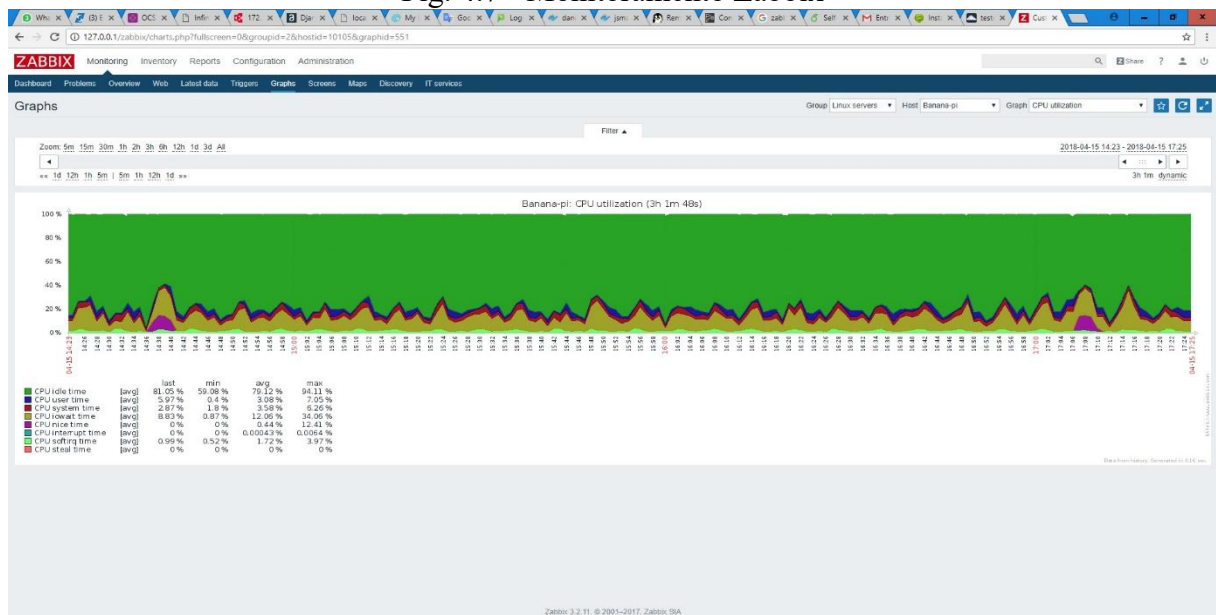
## 4.6 Zabbix

Zabbix é um *software* consolidado como ferramenta de monitoramento em redes de computadores, servidores e serviços. O mesmo possui o intuito de monitorar a integridade, disponibilidade, experiência de usuário e qualidade de serviços.

Segundo Dalle Vacche e Kewan Lee (2015), o Zabbix surgiu em 2001 e desde o seu lançamento se distinguiu como uma solução de monitoramento poderosa e eficaz. Trata-se de um produto de código aberto, fácil de obter e implementar. Sua abordagem possui métricas e alarmes que ajudam bastante o administrador de redes.

Fornece ótimos relatórios, assim como a possibilidade de visualizar dados de recursos com base nas informações armazenadas. Possui ainda um excelente mecanismo de notificação, o qual possibilita aos usuários configurar alertas para qualquer evento. A figura 4.7 demonstra um monitoramento de uso da CPU de um dispositivo embarcado realizado com o *software* Zabbix.

Fig. 4.7 - Monitoramento Zabbix



Fonte: Própria do autor (2018)



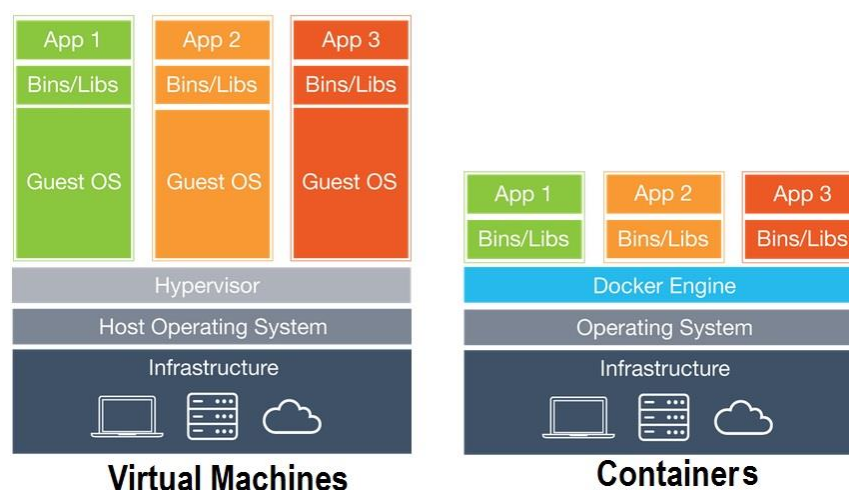
O Zabbix é uma solução ideal para pequenos e grandes ambientes distribuídos, onde é possível gerenciar de forma eficiente e extrair informações significativas de objetos e eventos.

## 4.7 Docker Container

A tecnologia Docker, é uma implementação da tecnologia "virtualização em nível de contêiner". No mecanismo do Docker, cada contêiner tem seus próprios *namespaces* independentes, que são transparentes e isolados de outros. Além disso, o mecanismo Docker pode gerenciar efetivamente os recursos do sistema operacional, dividindo-os em grupos separados para equilibrar ainda mais as necessidades de recursos conflitantes (GENG, 2017).

Docker não é um sistema de virtualização tradicional. Enquanto em um ambiente de virtualização tradicional nos nós temos um S.O. completo e isolado, dentro do Docker nos nós temos recursos isolados que utilizando bibliotecas de *kernel* em comum (entre *host* e *container*), isso é possível pois o Docker utiliza como *backend* o nosso conhecido LXC. A figura 4.8 demonstra a diferença entre a implementação tradicional (VM) e a *container* (Docker Engine).

Fig. 4.8 - Diferença entre VM e Docker



Fonte: <https://talkitbr.com> (2018)

O Docker é uma solução para containerização, que é uma alternativa para as tradicionais máquinas virtuais. Tanto máquinas virtuais como *containers* proporcionam recursos computacionais, tendo como benefícios o isolamento e facilidade de alocação de ambiente computacional. Contudo, os containers têm uma arquitetura diferente. Enquanto as máquinas virtuais têm um sistema operacional completo e vários *softwares* e bibliotecas para executar as

aplicações, os *containers* possuem apenas as aplicações e as bibliotecas dependentes, compartilhando uma mesma *engine*. Consequentemente, os *containers* são bem menores e podem ser facilmente compartilhados e replicados. Possibilitando assim, a implementação em plataformas de sistemas embarcados de baixo recursos computacionais, como é o caso do Raspberry Pi 3 modelo B.

O Docker possibilita o empacotamento de uma aplicação ou ambiente inteiro dentro de um *container*, e a partir desse momento o ambiente inteiro torna-se portátil para qualquer outro *host* que contenha o Docker instalado.

### AVALIAÇÃO E RESULTADOS

Este capítulo apresenta a implementação, análise e avaliação com o *benchmark* Smashbox, simulando um ambiente de *Fog Computing* para validar o desempenho do serviço de armazenamento StaaS (*Storage as a Service*), implementando o sistema de nuvem OwnCloud nas plataformas embarcadas de sistemas embarcados Raspberry PI, Banana PI e o servidor DELL PowerEdge T410. Essa etapa consiste em: realizar a montagem do cenário de teste com os dispositivos de sistemas embarcados e o servidor de forma individual; efetuar a abordagem dos *softwares* necessários nos dispositivos para a elaboração do experimento; proceder com o processo de coleta dos dados do tempo de *upload*, velocidade do *upload*, tempo de *download*, velocidade do *download* e tempo total de cada teste realizado com o *software benchmark* Smashbox, com a finalidade de gerar cargas e executar elevados fluxos de dados na disponibilização do serviço de armazenamento StaaS (*Storage as a Service*), e em paralelo realizar o monitoramento do consumo de processamento, memória e rede; e por fim, aferir os resultados.

#### 5.1 Elaboração do Cenário de Testes

Ao realizar a montagem do cenário para elaboração dos testes, foi necessário instalar um sistema operacional nos dispositivos embarcados e no servidor DELL PowerEdge T410. Para ser realizada uma análise do impacto no fornecimento do serviço de armazenamento StaaS (*Storage as a Service*), foi implementado 5 (cinco) cenários com diferentes sistemas operacionais, com e sem a implementação da virtualização Docker *Container*, isso possibilitou analisar o melhor conjunto implementado e o impacto da virtualização em dispositivos de sistemas embarcados.



Para isso utilizou-se o *Ubuntu Server* no Banana Pi M3, não foi possível implementar a virtualização *Docker Container* no Banana Pi M3, pois o mesmo só é capaz de utilizar o *Kernel* até a versão 3.4 do sistema operacional Linux e para poder ser implementada a virtualização *Docker Container* a versão do *Kernel* tem que ser 3.10 ou superior. Já no Raspberry Pi 3, foram instaladas três diferentes implementações, o Raspbian *Stretch Lite* sem virtualização *Docker Container* e com virtualização, também foi testada a implementação do Raspberry Pi 3 com o sistema operacional Ubuntu Mate com virtualização *Docker*, o único com interface gráfica. Em sequência, ocorreu a instalação do *software* de nuvem Owncloud na versão 10 para fornecer o serviço de armazenamento StaaS (*Storage as a Service*). O quadro 5.1 mostra as diferentes implementações utilizadas com diferentes sistemas operacionais, uso da virtualização *Docker Container* ou não, capacidade da memória RAM e capacidade de transmissão da placa de rede, em cada equipamento utilizado.

Quadro 5.1 - Diferentes implementações de sistemas operacionais

Equipamentos utilizados na avaliação				
EQUIPAMENTO	S.O	VIRTUALIZAÇÃO	RAM	NETWORK
Banana PI M3	Ubuntu Server 16.04	Não suporta	2 Gb	1000 Mb/s
Raspberry PI 3	Raspbian Stretch Lite 9	Sem virtualização	1 Gb	100 Mb/s
Raspberry PI 3	Raspbian Stretch Lite 9	Docker	1 Gb	100 Mb/s
Raspberry PI 3	Ubuntu MATE 16.04	Docker	1 Gb	100 Mb/s
Servidor Dell T410	Ubuntu Server 16.04	VMware ESXi + Docker	16 Gb	1000 Mb/s
Desktop Dell OptiPlex 5050	Windows 10	Docker	16 Gb	1000 Mb/s

Fonte: Própria do autor (2018)

Ocorreu também a instalação *Docker Container* para o sistema Windows 10 no Desktop Dell OptiPlex 5050 e dois *containers* de aplicações. Um com o *software* de *benchmark* Smashbox e no outro o *software* de monitoramento Zabbix. Para a instalação desses *containers*, utilizou-se o sistema de virtualização *Docker*. No Servidor Dell T410, ocorreu também a instalação do sistema de virtualização VMware ESXi 6, assim como uma VM com o sistema operacional *Ubuntu Server* 16.04 com suporte ao sistema de virtualização *Docker Container* e

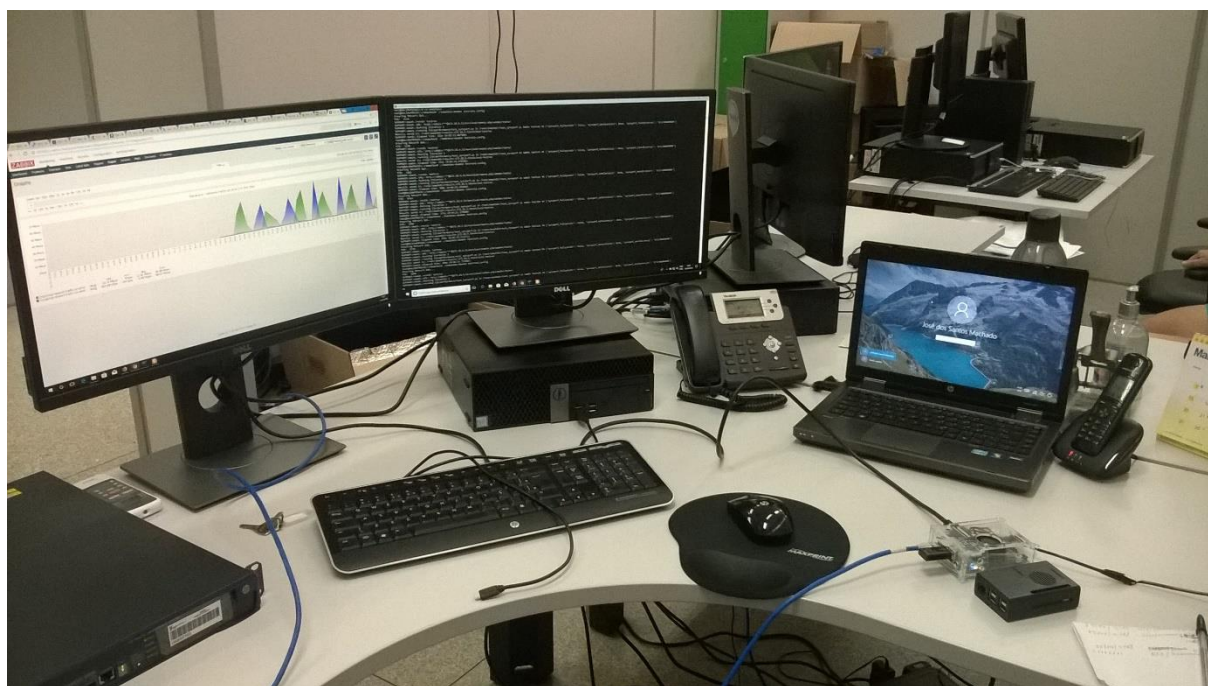
por fim, a instalação do *container* do sistema de nuvem OwnCloud na versão 10 para fornecer o serviço de armazenamento StaaS (*Storage as a Service*).

Foi preciso instalar dissipadores de calor, assim como um cooler, a fim de refrigerar os dispositivos, pois houve um elevado número de ocorrências de mensagens de alarme referindo-se a alta temperatura nos dispositivos de sistemas embarcados.

Para validar o cenário de testes, o sistema Smashbox foi configurado com a finalidade de executar a transferência e análise dos arquivos de cada teste. Durante os testes foi realizado o monitoramento do processamento, memória e volume do fluxo de dados na placa de rede, com o *software* Zabbix.

A intenção foi realizar uma análise de desempenho entre dois dispositivos de sistemas embarcados e um servidor de alto desempenho para fornecer o serviço de armazenamento StaaS (*Storage as a Service*), com uso do *software* de nuvem OwnCloud 10, usando-se do sistema de *benchmark* Smashbox, a qual consiste em: realizar a coleta do tempo utilizado e velocidade de *upload*; tempo utilizado e velocidade de *download*; apurar o tempo utilizado em cada teste; e em paralelo realizar medições do consumo do processamento, memória RAM e fluxo de dados na rede com o sistema de monitoramento de redes Zabbix. A figura 5.1 ilustra o cenário real em que os testes foram realizados.

Fig. 5.1 - Cenário de teste



Fonte: Própria do autor (2018)

### 5.1.1 Configuração do Smashbox

Foi utilizado um *container* da aplicação Smashbox disponibilizado por MRÓWCZYŃSKI (2017), também se utilizou do seu método de configuração, inicialmente foi feita atualização da aplicação do *client* do servidor OwnCloud para a versão mais recente disponível, em seguida foi configurado o arquivo `testrun.config`, onde encontra-se as variáveis de configuração do teste, sendo que o sistema Smashbox não tem interface gráfica, toda sua configuração é através de variáveis. Em um único teste o sistema Smashbox testa o *upload* e *download*, e todos os testes foram configurados para serem realizados com trinta repetições consecutivas, só foram aceitos para análise os testes concluídos sem erros. A figura 5.2 ilustra um exemplo de configuração do arquivo `testrun.config`.

Fig. 5.2 - Configuração do arquivo `testrun.config`

```
{
  "config" : [
    "remote=true",
    "sniffer=true",
    "backuplog=true",
    "remote_storage_server=YOUR_SERVER",
    "remote_database=YOUR_DB",
    "remote_storage_user=YOUR_DB_USR",
    "remote_storage_password=YOUR_DB_PSWD",
  ],
  [
    "engine=owncloud",
    "oc_server=YOUR_SERVER",
    "oc_account_name=YOUR_ACC",
    "oc_account_password=YOUR_PSW",
    "oc_server_folder=YOUR_REMOTE_FOLDER",
    "oc_sync_cmd=YOUR_CMD_DIR",
    "oc_webdav_endpoint=YOUR_WEBDAV",
    "oc_account_reset_procedure=webdav_delete",
    "oc_ssl_enabled=true",
  ],
],
  "tests" : [
    {
      "runid" : "testrun",
      "test_name" : "performance/test_syncperf.py",
      "testset" : "0"
    },
  ],
  "loop" : 1,
  "ensure_net_qos" : 10,
  "timeout" : 3600,
}
```

Fonte: MRÓWCZYŃSKI (2017)

O *benchmark* Smashbox é uma estrutura desenvolvida em python para testar de ponta a ponta a funcionalidade do serviço de armazenamento em nuvem. Essa estrutura de teste pode ser executada interativamente a partir de uma linha de comando, executar testes contínuos por meio de tarefas agendadas ou testes de carga / estresse. Pode ser facilmente integrado em testes e processos de controle de qualidade (MRÓWCZYŃSKI, 2017).

O que foi verificado no Smashbox:

- Sincronização cliente em vários cenários;
- Compartilhamento de arquivos e pastas;
- Verificação de conflitos de protocolos;
- Tempo utilizado e velocidade de *upload*;
- Tempo utilizado e velocidade de *download*;
- Tempo utilizado em cada teste.

O que foi verificado no Zabbix:

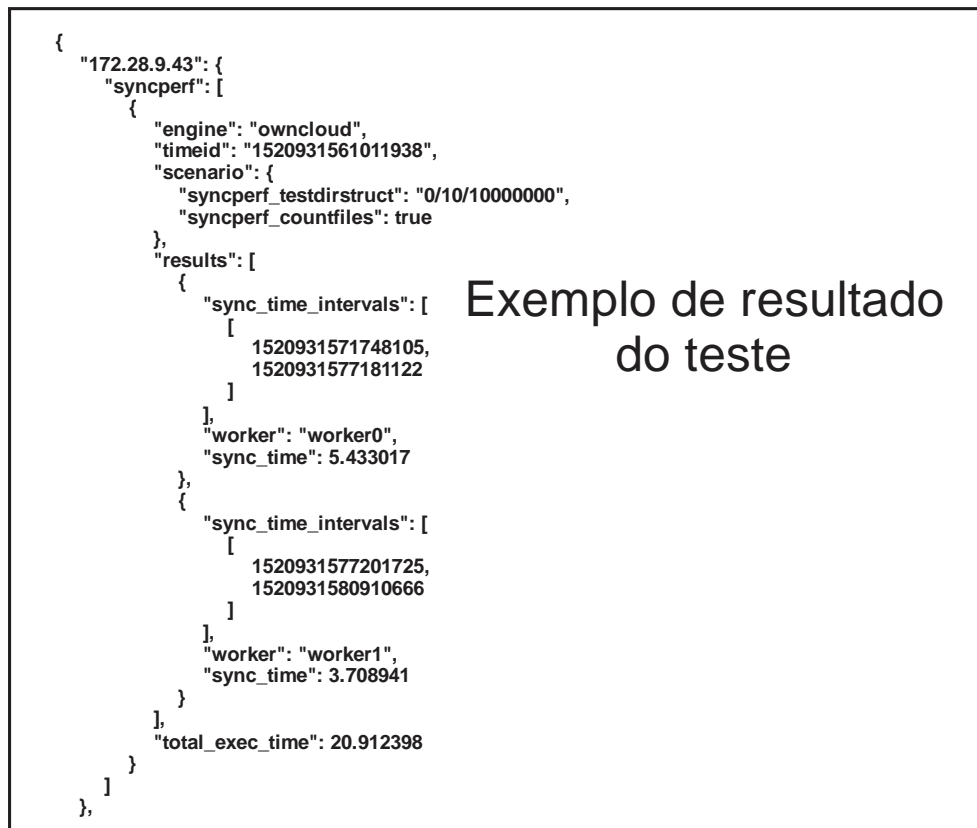
- Consumo do processamento;
- Uso de memória RAM;
- Fluxo de dados de *upload* na rede;
- Fluxo de dados de *download* na rede.

## 5.2 Análise e Avaliação dos Resultados

Após a realização dos testes preliminares para validar o cenário que objetiva simular um ambiente real de produção, iniciou-se a coleta do tempo e velocidade utilizado para *upload*, tempo e velocidade utilizado para *download*, tempo total necessário para cada teste, consumo de processamento, uso de memória RAM e o fluxo de dados de *upload* e *download* na placa de rede de cada equipamento testado.

Dessa forma, foi possível realizar a captura dos dados e consequentemente efetuar a análise. Todavia, totalizaram-se 750 coletas de dados (30 repetições para os 5 diferentes testes, utilizando as 5 diferentes implementações entre os equipamentos testados e diferentes sistemas operacionais). Assim, foi possível obter um bom quantitativo de dados para realizar a análise. A figura 5.3 mostra um exemplo de resultado obtido pelo sistema de *benchmark* Smashbox.

Fig. 5.3 - Resultado de teste com Smashbox



Fonte: Própria do autor (2018)

Onde-se lê as variáveis "syncperf\_testdirstruct" indica o tipo do teste realizado, "worker0" o tempo de *upload*, "worker1" o tempo de *download* e "total\_exec\_time" o tempo total da realização do teste. Contudo, utilizou-se a média, desvio padrão e intervalo de confiança de 95% como parâmetro para as medições. No entanto, os cálculos foram realizados manualmente, através da exportação dos dados capturados para o sistema Microsoft Office Excel.

As subseções seguintes apresentam os resultados coletados com o sistema de *benchmark* Smashbox e o sistema de monitoramento de rede Zabbix nos dispositivos de sistemas embarcados Raspberry Pi 3, Banana Pi M3 e no servidor DELL PowerEdge T410, com o seu respectivo desvio padrão e intervalo de confiança de 95%, assim como breves discussões.

### 5.2.1 Test0 0/1/1

O test0 0/1/1 consiste no menor volume testado nesse trabalho para a transferência de arquivos em armazenamento em nuvem, esse teste avalia a transferência de um arquivo com o

tamanho de 1 byte, é ideal para se analisar como o sistema se comporta com a transferência de um pequeno arquivo. A tabela 5.1 mostra os resultados coletados do tempo médio utilizado e velocidade de *upload*; tempo médio utilizado e velocidade de *download*; apura o tempo médio utilizado em cada teste; o desvio padrão, o valor mínimo e máximo obtidos nos resultados com o sistema de *benchmark* Smashbox e em paralelo apresenta as medições do máximo atingido do consumo do processamento, memória RAM e fluxo de dados na rede com o sistema de monitoramento de rede Zabbix em cada implementação testada, o tempo encontra-se em segundos.

Tabela 5.1 - Resultados do Test0 - 1 x 1B = 1 Byte

BANANA PI M3 + UBUNTU SERVER								
TEST0 0/1/1 1 x 1B = 1 Byte	SMASHBOX				ZABBIX			
	MÉDIA	DESV	MÍN	MAX	CPU %	MEM %	UPL kbps	DOW kbps
Tempo de <i>upload</i> em segundos	15,13	5,54	3,00	26,00	21,50	12,50	6,59	10,02
Velocidade <i>upload</i> em bps	0,08	0,06	0,04	0,33				
Tempo <i>download</i> em segundos	3,33	2,01	2,00	9,00				
Velocidade <i>download</i> em bps	0,38	0,15	0,11	0,50				
Tempo do teste em segundos	38,23	8,65	21,00	57,00				
RASPBERRY PI 3 + RASPBIAN STRETCH LITE								
Tempo de <i>upload</i> em segundos	4,20	5,26	1,00	17,00	32,42	31,39	8,99	18,34
Velocidade <i>upload</i> em bps	0,73	0,42	0,06	1,00				
Tempo <i>download</i> em segundos	1,73	2,30	0,82	13,00				
Velocidade <i>download</i> em bps	0,93	0,38	0,08	1,22				
Tempo do teste em segundos	20,40	22,23	6,00	62,00				
RASPBERRY PI 3 + RASPBIAN STRETCH LITE + DOCKER								
Tempo de <i>upload</i> em segundos	22,60	3,76	14,00	31,00	29,36	26,55	4,52	6,97
Velocidade <i>upload</i> em bps	0,05	0,01	0,03	0,07				
Tempo <i>download</i> em segundos	14,53	2,61	7,00	20,00				
Velocidade <i>download</i> em bps	0,07	0,02	0,05	0,14				
Tempo do teste em segundos	106,60	10,41	73,00	122,00				
RASPBERRY PI 3 + UBUNTU MATE + DOCKER								
Tempo de <i>upload</i> em segundos	12,83	1,64	9,00	17,00	49,93	41,09	4,67	2,57
Velocidade <i>upload</i> em bps	0,08	0,01	0,06	0,11				
Tempo <i>download</i> em segundos	7,03	1,27	4,00	11,00				
Velocidade <i>download</i> em bps	0,15	0,03	0,09	0,25				
Tempo do teste em segundos	60,87	2,15	56,00	64,00				
SERVIDOR DELL T410 + UBUNTU SERVER + VMWARE ESXI + DOCKER								
Tempo de <i>upload</i> em segundos	1,00	0,00	1,00	1,00	6,40	6,00	8,77	15,52
Velocidade <i>upload</i> em bps	1,00	0,00	1,00	1,00				
Tempo <i>download</i> em segundos	0,99	0,02	0,90	1,00				
Velocidade <i>download</i> em bps	1,01	0,02	1,00	1,11				
Tempo do teste em segundos	6,00	0,26	5,00	7,00				

Fonte: Própria do autor (2018)

Nota-se que nesse teste todas as implementações obtiveram resultados melhores no tempo de *download* em relação ao tempo de *upload* e consequentemente maior velocidade.

Também se observa que as implementações nos dispositivos de sistemas embarcados obtiveram um valor muito elevado na métrica de desvio padrão, que será analisado nos testes seguintes. Para melhor entendimento dos resultados foi realizada uma análise resumida de cada implementação separadamente a seguir:

**BANANA PI M3 + UBUNTU SERVER:** Esta implementação deixou a desejar, já que o dispositivo Banana Pi M3 tem elevado poder de recursos (processamento, memória e rede) comparado com o seu concorrente Raspberry Pi 3, na métrica de tempo de *upload* só foi melhor do que a implementação do Raspberry com raspbian e virtualização Docker, e seus resultados ficaram bem abaixo comparado com a implementação do Raspberry com raspbian sem virtualização. Todavia obteve um baixo uso de processamento e memória durante os testes comparados com as outras implementações do Raspberry. Não foi possível implementar a virtualização neste dispositivo pelo fato de seu *hardware* não suportar.

**RASPBERRY PI 3 + RASPBIAN STRETCH LITE:** Esta implementação surpreendeu pelo seu desempenho, chegando a obter melhor resultado em relação ao servidor DELL T410, foi o caso do parâmetro tempo mínimo obtido no *download* 0,82 segundos sendo a melhor performance entre todas as implementações, obteve também a maior velocidade de *download* 18,34kbps verificada pelo monitoramento da rede com o sistema Zabbix. Todavia nota-se que o desvio padrão dos seus resultados apresentaram uma alta discrepância.

**RASPBERRY PI 3 + RASPBIAN STRETCH LITE + DOCKER:** Nesta implementação nota-se que a utilização da virtualização Docker *container* em um único nó onera bastante o desempenho do dispositivo, já que o sistema operacional deve tratar uma camada a mais na sua implementação. Observa-se que foi a implementação que obteve os piores resultados, com a média 22,60 segundos de *upload* e 14,53 segundos de *download*, e uma média de tempo de conclusão dos testes muito elevada 106,60 segundos e com um nível de uso de processamento e memória mediano.

**RASPBERRY PI 3 + UBUNTU MATE + DOCKER:** Esta implementação é a única que o sistema operacional foi disponibilizado com um ambiente de *interface* gráfica e obteve bons resultados, chegando a obter melhor resultado quando comparado com a implementação do Banana Pi M3 utilizando o sistema ubuntu *server* sem virtualização, foi o caso da métrica do tempo médio de *upload* com 12,83 segundos contra 15,13 segundos do seu concorrente.

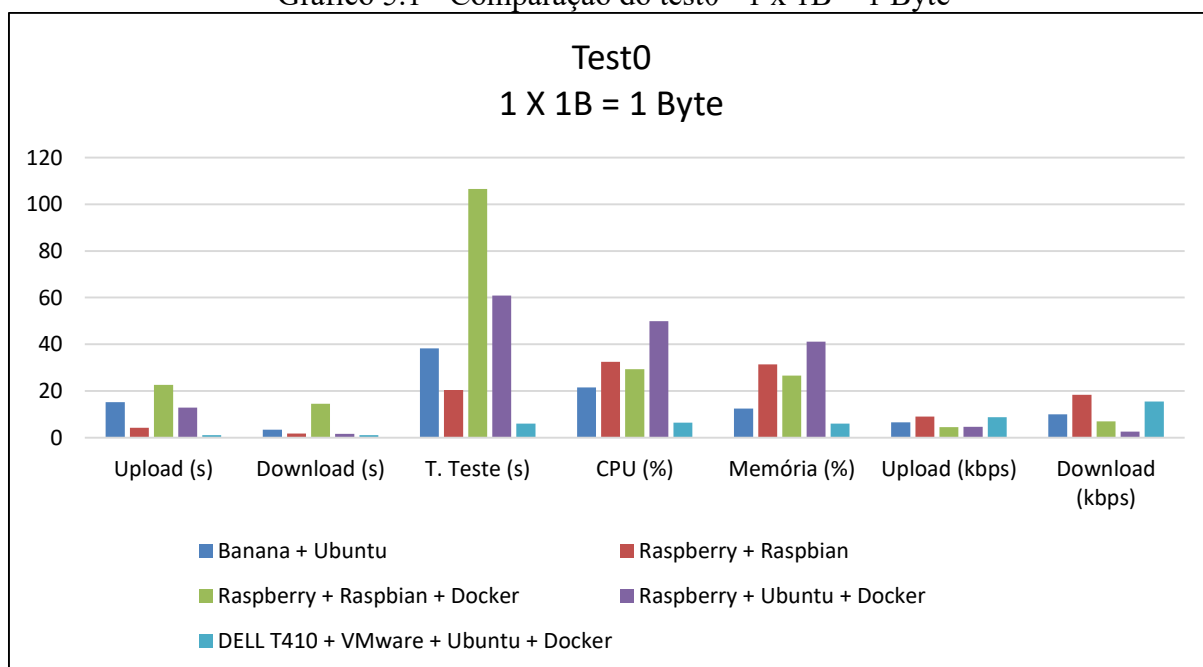
Todavia nota-se que foi a implementação que mais se utilizou do processamento e memória do *hardware*.

#### **SERVIDOR DELL T410 + UBUNTU SERVER + VMWARE ESXI + DOCKER:**

Está implementação é considerada a ideal, afinal o servidor DELL PowerEdge T410 é um equipamento de alto desempenho, porém com um alto custo aquisitivo, percebe-se que os seus resultados foram melhores em quase todos os aspectos analisados em relação às outras implementações. Nota-se que o tempo de *upload* e *download* praticamente são os mesmos, o que não acontece com os dispositivos de sistemas embarcados, e suas métricas relacionadas ao desvio padrão dos resultados obtidos são praticamente zerados nas 30 repetições do teste, isso indica que o serviço é bem mais preciso. Outro dado interessante de ser analisado é a métrica média de tempo total do teste, que inclui *upload*, *download* e checagem dos dados que obteve 6,00 segundos, esse resultado é mais de três vezes melhor em relação ao segundo colocado 20,40 obtido pela implementação Raspberry com raspbian *stretch lite* sem virtualização. Isso com o pico de uso do processamento e memória muito baixo.

O gráfico 5.1 mostra melhor visualmente a comparação entre todas as implementações referentes ao tempo médio de *upload*, tempo médio de *download* e tempo médio utilizado na execução dos testes, obtidos pelo sistema de *benchmark* Smashbox, também apresenta os valores do pico de uso do processamento, memória, *upload* e *download*, capturados com o sistema de monitoramento de rede Zabbix.

Gráfico 5.1 - Comparação do test0 - 1 x 1B = 1 Byte



Fonte: Própria do autor (2018)



### 5.2.2 Test1 0/1/100000000

O test1 0/1/100000000 consiste em volume mediano testado nesse trabalho para a transferência de arquivos em armazenamento em nuvem, esse teste avalia a transferência de um arquivo com o tamanho de 100 Megabytes, é ideal para se analisar como o sistema se comporta com a transferência de um arquivo com volume mediano. A tabela 5.2 mostra os resultados coletados do tempo médio utilizado e velocidade de *upload*; tempo médio utilizado e velocidade de *download*; apurar o tempo médio utilizado em cada teste; o desvio padrão, o valor mínimo e máximo obtidos nos resultados com o sistema de *benchmark* Smashbox e em paralelo demonstra as medições do máximo atingido do consumo do processamento, memória RAM e fluxo de dados na rede com o sistema de monitoramento de rede Zabbix em cada implementação testada, o tempo encontra-se em segundos.

Tabela 5.2 - Resultados do Test1 - 1 x 100Mb = 100 Megabytes

BANANA PI M3 + UBUNTU SERVER									
TEST1 0/1/100000000 1 x 100Mb = 100 Megabytes	SMASHBOX				ZABBIX				
	MÉD	DES	MÍN	MAX	CPU %	MEM %	UPL mbps	DOW mbps	
	Tempo de <i>upload</i> em segundos	88,50	14,45	67,00	125,00	36,31	15,50	19,80	14,08
	Velocidade <i>upload</i> em mbps	1,16	0,17	0,80	1,49				
	Tempo <i>download</i> em segundos	10,60	4,58	6,00	27,00				
	Velocidade <i>download</i> em mbps	10,62	3,12	3,70	16,70				
	Tempo do teste em segundos	121,63	19,24	95,00	159,00				
RASPBERRY PI 3 + RASPBIAN STRETCH LITE									
Tempo de <i>upload</i> em segundos	56,93	26,51	30,00	144,00	56,21	33,06	19,66	14,86	
Velocidade <i>upload</i> em mbps	2,06	0,74	0,69	3,33					
Tempo <i>download</i> em segundos	9,70	1,78	9,00	18,00					
Velocidade <i>download</i> em mbps	10,53	1,26	5,56	11,11					
Tempo do teste em segundos	77,67	34,84	44,00	191,00					
RASPBERRY PI 3 + RASPBIAN STRETCH LITE + DOCKER									
Tempo de <i>upload</i> em segundos	199,20	16,61	148,00	223,00	67,60	28,76	17,50	14,17	
Velocidade <i>upload</i> em mbps	0,51	0,05	0,45	0,68					
Tempo <i>download</i> em segundos	23,93	5,56	14,00	42,00					
Velocidade <i>download</i> em mbps	4,39	1,02	2,38	7,14					
Tempo do teste em segundos	290,90	25,80	174,00	317,00					
RASPBERRY PI 3 + UBUNTU MATE + DOCKER									
Tempo de <i>upload</i> em segundos	125,13	7,29	117,00	153,00	55,96	40,43	19,38	14,12	
Velocidade <i>upload</i> em mbps	0,80	0,04	0,65	0,85					
Tempo <i>download</i> em segundos	23,57	4,28	17,00	32,00					
Velocidade <i>download</i> em mbps	4,39	0,83	3,13	5,88					
Tempo do teste em segundos	197,13	9,67	184,00	237,00					
SERVIDOR DELL T410 + UBUNTU SERVER + VMWARE ESXI + DOCKER									
Tempo de <i>upload</i> em segundos	9,23	0,43	9,00	10,00	14,03	6,00	40,31	28,78	
Velocidade <i>upload</i> em mbps	10,85	0,48	10,00	11,11					
Tempo <i>download</i> em segundos	4,00	0	4,00	4,00					
Velocidade <i>download</i> em mbps	25,00	0	25,00	25,00					
Tempo do teste em segundos	18,87	0,35	18,00	19,00					

Fonte: Própria do autor (2018)

Nota-se que nesse teste todas as implementações obtiveram resultados melhores no tempo de *download* em relação ao tempo de *upload* e consequentemente maior velocidade. Também se observa que as implementações nos dispositivos de sistemas embarcados obtiveram um valor muito elevado na métrica de desvio padrão. Para melhor entendimento dos resultados foi realizada uma análise resumida de cada implementação separadamente a seguir:

**BANANA PI M3 + UBUNTU SERVER:** Neste teste essa implementação obteve um melhor resultado em relação ao teste anterior, obteve uma métrica melhor no quesito tempo médio de *upload* comparado a implementação do Raspberry com ubuntu e utilizando virtualização Docker, fato que não tinha ocorrido no teste Test0. Porém, seus resultados ficaram abaixo comparado com a implementação do Raspberry com raspbian sem virtualização. Todavia obteve um baixo uso de processamento e memória durante os testes em relação às outras implementações do Raspberry.

**RASPBERRY PI 3 + RASPBIAN STRETCH LITE:** Esta implementação continuou obtendo bons resultados, perdendo apenas para a implementação do servidor DELL T410. Todavia nota-se que o desvio padrão dos seus resultados apresentaram uma alta discrepância, obteve na métrica de desvio padrão do tempo médio utilizado para cada teste o resultado de 34,84 segundos, melhor e pior tempo obtido neste parâmetro 44,00 e 191,00 segundos respectivamente, isso o torna um serviço pouco preciso. Percebe-se também que esta implementação obteve um alto uso de processamento e memória 55,96% e 40,43% respectivamente verificada pelo monitoramento da rede com o sistema Zabbix.

**RASPBERRY PI 3 + RASPBIAN STRETCH LITE + DOCKER:** Nesta implementação nota-se que continuou obtendo os piores resultados no teste de disponibilização do serviço StaaS (*Storage as a Service*), com a média 199,20 segundos de *upload* e 23,93 segundos de *download*, e uma média de tempo de conclusão dos testes muito elevada 290,90 segundos. Percebe-se também que foi a implementação que chegou ao mais alto nível de processamento no teste 67,60% em relação a todas as outras implementações e com o uso de memória mediano 28,76%.

**RASPBERRY PI 3 + UBUNTU MATE + DOCKER:** Esta implementação obteve bons resultados quando comparado com a implementação de concorrência direta, Raspberry Pi com Raspbian Stretch e utilizando a virtualização Docker *Container*, foi o caso das métricas do

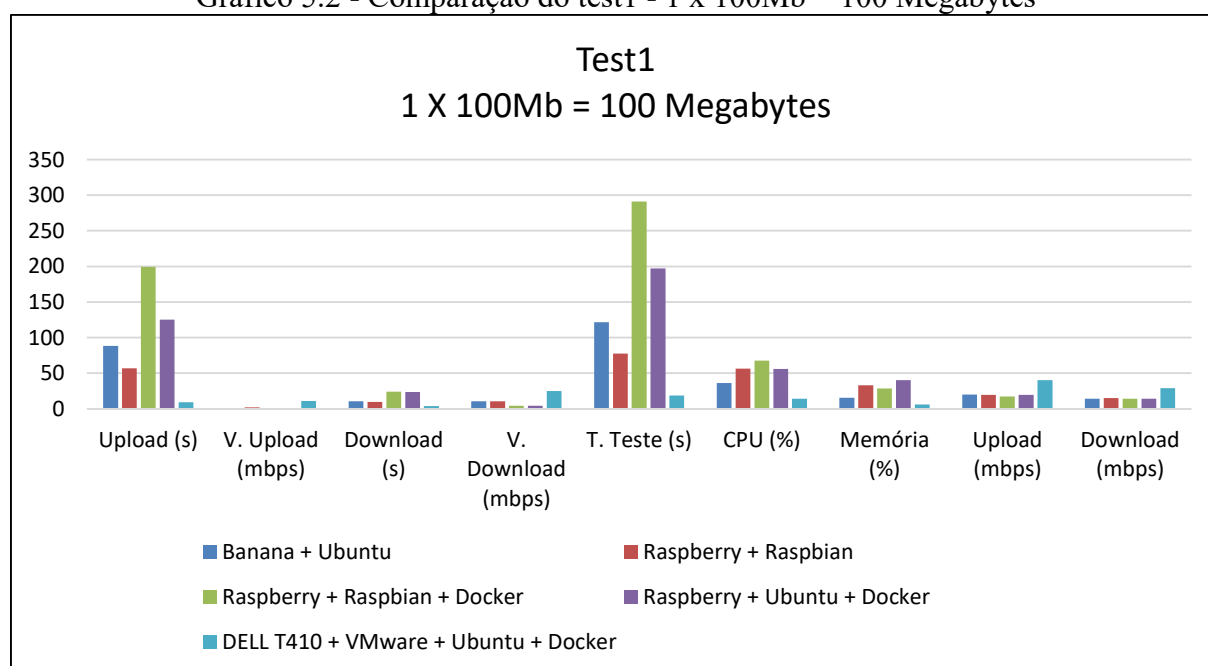
tempo médio de *upload*, tempo médio de *download* e tempo médio utilizado para a realização dos testes, 125,13 segundos, 23,57 segundos e 197,13 segundos respectivamente. Todavia nota-se que foi a implementação que obteve o mais alto nível do uso da memória do *hardware* 40,43%.

### SERVIDOR DELL T410 + UBUNTU SERVER + VMWARE ESXI + DOCKER:

Percebe-se que nesse teste essa implementação obteve os melhores resultados em todos os aspectos analisados em relação às outras implementações. Nota-se que o tempo médio de *upload* e *download* são excelentes, sem grandes variações nas métricas mínimo e máximo obtidos nos resultados, isso demonstra um serviço bem preciso, o que não acontece com os dispositivos de sistemas embarcados, e suas métricas relacionadas ao desvio padrão dos resultados obtidos são muito baixas nas 30 repetições do teste. Outro dado interessante de ser analisado é a métrica referente a média de velocidade de *download* obtido pelo sistema de *benchmark* Smashbox 25Mbps, um excelente resultado comparado com as outras implementações. Isso com o pico de uso do processamento e memória muito baixo.

O gráfico 5.2 mostra melhor visualmente a comparação entre todas as implementações referentes ao tempo médio de *upload* e *download*, velocidade média de *upload* e *download*, e tempo médio utilizado na execução dos testes obtidos pelo sistema de *benchmark* Smashbox, também apresenta os valores do pico de uso do processamento, memória, velocidade de *upload* e *download*, capturados com o sistema de monitoramento de rede Zabbix.

Gráfico 5.2 - Comparação do test1 - 1 x 100Mb = 100 Megabytes



Fonte: Própria do autor (2018)

### 5.2.3 Test2 0/10/10000000

O test2 0/10/10000000 consiste em volume mediano testado nesse trabalho para a transferência de arquivos em armazenamento em nuvem, porém dividido em dez arquivos, esse teste avalia a transferência de dez arquivos com o tamanho de 10 Megabytes, totalizando 100 Megabytes de transferência, é ideal para se analisar como o sistema se comporta com a transferência de 10 arquivos com volume mediano de 10 Megabytes cada. A tabela 5.3 mostra os resultados coletados do tempo médio utilizado e velocidade de *upload*; tempo médio utilizado e velocidade de *download*; apurar o tempo médio utilizado em cada teste; o desvio padrão, o valor mínimo e máximo obtidos nos resultados com o sistema de *benchmark* Smashbox e em paralelo apresenta as medições do máximo atingido do consumo do processamento, memória RAM e fluxo de dados na rede com o sistema de monitoramento de rede Zabbix em cada implementação testada, o tempo encontra-se em segundos.

Tabela 5.3 - Resultados do Test2 - 10 x 10Mb = 100 Megabytes

BANANA PI M3 + UBUNTU SERVER									
TEST2 0/10/10000000 10 x 10Mb = 100 Megabytes	SMASHBOX				ZABBIX				
	MÉD	DES	MÍN	MAX	CPU %	MEM %	UPL mbps	DOW mbps	
	Tempo de <i>upload</i> em segundos	70,83	13,28	49,00	111,00	34,91	15,50	20,19	14,48
	Velocidade <i>upload</i> em mbps	1,46	0,26	0,90	2,04				
	Tempo <i>download</i> em segundos	11,77	7,54	5,00	32,00				
	Velocidade <i>download</i> em mbps	11,39	5,15	3,12	20,00				
	Tempo do teste em segundos	111,17	18,20	85,00	147,00				
RASPBERRY PI 3 + RASPBIAN STRETCH LITE									
Tempo de <i>upload</i> em segundos	30,63	29,74	16,00	158,00	40,88	33,85	24,24	20,03	
Velocidade <i>upload</i> em mbps	4,40	1,41	0,63	6,25					
Tempo <i>download</i> em segundos	10,20	3,92	9,00	30,00					
Velocidade <i>download</i> em mbps	10,38	1,65	3,33	11,11					
Tempo do teste em segundos	49,83	34,27	33,00	199,00					
RASPBERRY PI 3 + RASPBIAN STRETCH LITE + DOCKER									
Tempo de <i>upload</i> em segundos	209,63	14,72	184,00	236,00	41,13	30,37	10,43	13,97	
Velocidade <i>upload</i> em mbps	0,48	0,03	0,42	0,54					
Tempo <i>download</i> em segundos	59,37	8,273	50,00	95,00					
Velocidade <i>download</i> em mbps	1,71	0,18	1,05	2,00					
Tempo do teste em segundos	341,77	19,49	307,00	414,00					
RASPBERRY PI 3 + UBUNTU MATE + DOCKER									
Tempo de <i>upload</i> em segundos	133,37	21,92	117,00	242,00	41,96	39,50	11,71	14,13	
Velocidade <i>upload</i> em mbps	0,76	0,08	0,41	0,85					
Tempo <i>download</i> em segundos	42,40	7,96	34,00	80,00					
Velocidade <i>download</i> em mbps	2,41	0,31	1,25	2,94					
Tempo do teste em segundos	225,70	21,93	212,00	330,00					
SERVIDOR DELL T410 + UBUNTU SERVER + VMWARE ESXI + DOCKER									
Tempo de <i>upload</i> em segundos	7,40	0,89	6,00	9,00	13,20	6,00	40,08	28,54	
Velocidade <i>upload</i> em mbps	13,71	1,70	11,11	16,67					
Tempo <i>download</i> em segundos	4,80	0,61	4,00	6,00					
Velocidade <i>download</i> em mbps	21,17	2,74	16,67	25,00					
Tempo do teste em segundos	18,73	1,08	17,00	21,00					

Fonte: Própria do autor (2018)

Nota-se que nesse teste todas as implementações obtiveram resultados melhores no tempo de *download* em relação ao tempo de *upload* e consequentemente maior velocidade. Também se observa que as implementações nos dispositivos de sistemas embarcados obtiveram um valor muito elevado na métrica de desvio padrão. Para melhor entendimento dos resultados foi realizada uma análise resumida de cada implementação separadamente a seguir:

**BANANA PI M3 + UBUNTU SERVER:** Neste teste essa implementação obteve bons resultados quando comparados aos resultados obtidos pelas implementações do Raspberry com virtualização Docker, porém obteve piores resultados em quase todas as métricas comparados com seu concorrente direto, a implementação do Raspberry Pi 3 com o sistema raspbian e sem virtualização, obteve na métrica no quesito tempo médio de *upload* 70,83 segundos, isso é duas vezes maior do tempo obtido pela implementação do Raspberry com raspbian sem virtualização que obteve 30,63 segundo. Todavia obteve um baixo uso de processamento e memória durante os testes em relação as outras implementações do Raspberry.

**RASPBERRY PI 3 + RASPBIAN STRETCH LITE:** Esta implementação continuou obtendo bons resultados, perdendo apenas para a implementação do servidor DELL T410. Todavia nota-se que o desvio padrão dos seus resultados apresentaram uma alta discrepância, obteve na métrica de desvio padrão do tempo médio utilizado para cada teste o resultado de 34,27 segundos, melhor e pior tempo obtido neste parâmetro 33,00 e 199,00 segundos respectivamente, isso o torna um serviço pouco preciso. Percebe-se também que esta implementação obteve um uso mediano de processamento e memória 40,88% e 33,85% respectivamente verificada pelo monitoramento da rede com o sistema Zabbix.

**RASPBERRY PI 3 + RASPBIAN STRETCH LITE + DOCKER:** Nesta implementação nota-se que continuou obtendo os piores resultados no teste de disponibilização do serviço StaaS (*Storage as a Service*), com a média 209,63 segundos de *upload* e 59,37 segundos de *download*, e uma média de tempo de conclusão dos testes muito elevada 341,77 segundos. Percebe-se também que a implementação chegou a utilizar um nível razoável de processamento no teste 41,13% e com o uso de memória mediano 30,37%.

**RASPBERRY PI 3 + UBUNTU MATE + DOCKER:** Esta implementação continuou obtendo bons resultados quando comparado com a implementação de concorrência direta, Raspberry Pi com Raspbian Stretch e utilizando a virtualização Docker *Container*, foi o caso

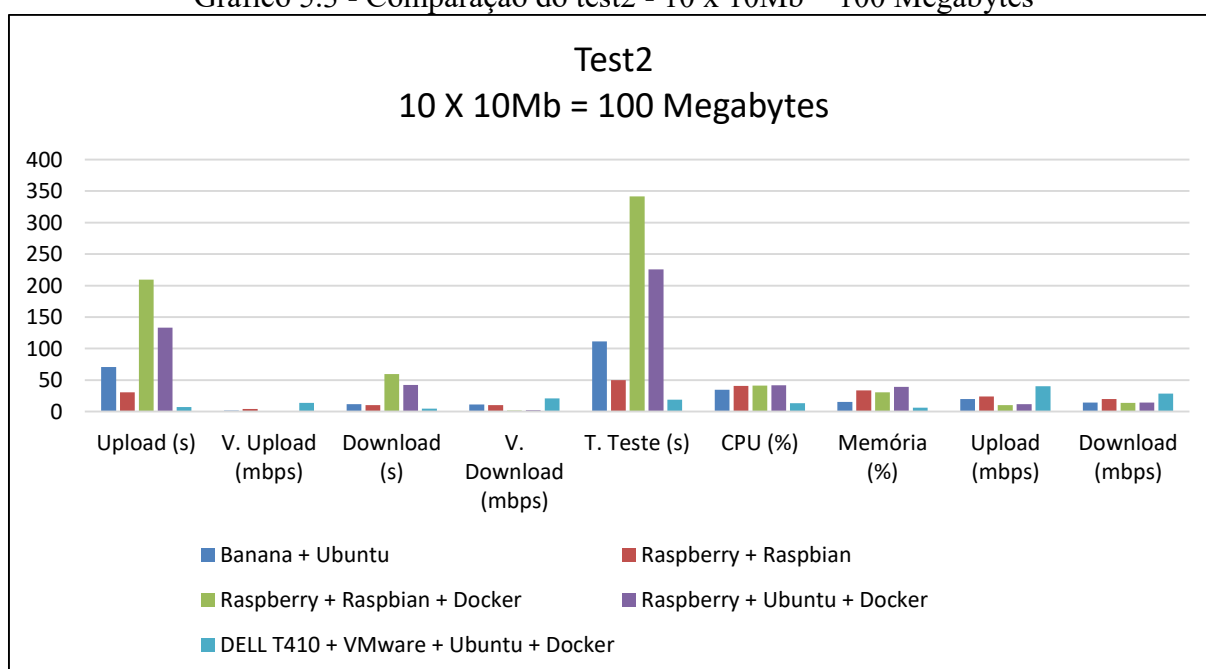
das métricas do tempo médio de *upload*, tempo médio de *download* e tempo médio utilizado para a realização dos testes, 133,37 segundos, 42,40 segundos e 225,70 segundos respectivamente. Todavia nota-se que foi a implementação que obteve o mais alto nível do uso de processamento e memória do *hardware* 41,96% e 39,50 respectivamente.

### SERVIDOR DELL T410 + UBUNTU SERVER + VMWARE ESXI + DOCKER:

Percebe-se que essa implementação continuo obtendo os melhores resultados em todos os aspectos analisados em relação às outras implementações. Nota-se que o tempo médio de 7,40 segundos para *upload* e de 4,80 segundos para *download* são excelentes, sem grandes variações nas métricas mínimo e máximo obtidos nos resultados, isso demonstra um serviço bem preciso, o que não acontece com os dispositivos de sistemas embarcados. Outros dados interessantes de serem analisados são as métricas referentes ao fluxo de dados de *upload* e *download* analisados na placa de rede do dispositivo com o sistema de monitoramento Zabbix, que registrou uma velocidade de 40,08Mbps e 28,58Mbps respectivamente, um excelente resultado comparado com as outras implementações.

O gráfico 5.3 mostra melhor visualmente a comparação entre todas as implementações referentes ao tempo médio de *upload* e *download*, velocidade média de *upload* e *download*, e tempo médio utilizado na execução dos testes obtidos pelo sistema de *benchmark* Smashbox, também apresenta os valores do pico de uso do processamento, memória, velocidade de *upload* e *download*, capturados com o sistema de monitoramento de rede Zabbix.

Gráfico 5.3 - Comparação do test2 - 10 x 10Mb = 100 Megabytes



Fonte: Própria do autor (2018)

### 5.2.4 Test3 0/1000/10000

O test3 0/1000/10000 consiste em um pequeno volume de dados testado nesse trabalho para a transferência de arquivos em armazenamento em nuvem, esse teste avalia a transferência de mil arquivos com o tamanho de 10 Kilobytes, totalizando 10 Megabytes de transferência, acredita-se que esse teste seja a simulação mais próxima da realidade do funcionamento de uma *Fog Computing*, pequenos arquivos, porém em grande quantidade. A tabela 5.4 mostra os resultados coletados do tempo médio utilizado e velocidade de *upload*; tempo médio utilizado e velocidade de *download*; apurar o tempo médio utilizado em cada teste; o desvio padrão, o valor mínimo e máximo obtidos nos resultados com o sistema de *benchmark* Smashbox e em paralelo apresenta as medições do máximo atingido do consumo do processamento, memória RAM e fluxo de dados na rede com o sistema de monitoramento de rede Zabbix em cada implementação testada, o tempo encontra-se em segundos.

Tabela 5.4 - Resultados do Test3 - 1000 x 10Kb = 10 Megabytes

BANANA PI M3 + UBUNTU SERVER								
TEST3 0/1000/10000 1000 x 10Kb = 10 Megabytes	SMASHBOX				ZABBIX			
	MÉD	DES	MÍN	MAX	CPU %	MEM %	UPL kbps	DOW kbps
Tempo de <i>upload</i> em segundos	640,67	459,94	418,00	2496,00	59,67	18,00	437,43	474,15
Velocidade <i>upload</i> em kbps	19,08	5,41	4,01	23,92				
Tempo <i>download</i> em segundos	301,67	121,19	232,00	687,00				
Velocidade <i>download</i> em kbps	36,04	7,65	14,56	43,10				
Tempo do teste em segundos	1079,03	540,16	804,00	2889,00				
RASPBERY PI 3 + RASPBIAN STRETCH LITE								
Tempo de <i>upload</i> em segundos	289,40	174,19	148,00	746,00	87,21	46,56	1000	1160
Velocidade <i>upload</i> em kbps	46,68	21,80	13,41	67,57				
Tempo <i>download</i> em segundos	120,77	98,14	87,00	546,00				
Velocidade <i>download</i> em kbps	101,22	25,48	18,31	114,94				
Tempo do teste em segundos	521,50	212,90	346,00	1175,00				
RASPBERY PI 3 + RASPBIAN STRETCH LITE + DOCKER								
Tempo de <i>upload</i> em segundos	369,03	46,41	346,00	610,00	67,65	31,14	423,69	381,18
Velocidade <i>upload</i> em kbps	27,36	2,18	16,39	28,90				
Tempo <i>download</i> em segundos	334,43	25,82	327,00	469,00				
Velocidade <i>download</i> em kbps	30,03	1,69	21,32	30,58				
Tempo do teste em segundos	812,60	56,57	779,00	1080,00				
RASPBERY PI 3 + UBUNTU MATE + DOCKER								
Tempo de <i>upload</i> em segundos	430,57	81,83	335,00	685,00	61,65	60,82	333,38	500,16
Velocidade <i>upload</i> em kbps	23,97	4,15	14,60	29,90				
Tempo <i>download</i> em segundos	278,13	210,58	106,00	1056,00				
Velocidade <i>download</i> em kbps	58,02	34,63	9,47	94,30				
Tempo do teste em segundos	821,17	292,95	553,00	1858,00				
SERVIDOR DELL T410 + UBUNTU SERVER + VMWARE ESXI + DOCKER								
Tempo de <i>upload</i> em segundos	350,63	6,14	340,00	364,00	12,00	7,08	100,70	135,90
Velocidade <i>upload</i> em kbps	28,53	0,50	27,47	29,41				
Tempo <i>download</i> em segundos	163,93	0,25	163,00	164,00				
Velocidade <i>download</i> em kbps	61,00	0,09	60,98	61,35				
Tempo do teste em segundos	620,63	6,36	610,00	635,00				

Fonte: Própria do autor (2018)

Nota-se que nesse teste todas as implementações obtiveram resultados melhores no tempo de *download* em relação ao tempo de *upload* e consequentemente maior velocidade. Também se observa que as implementações nos dispositivos de sistemas embarcados obtiveram um valor muito elevado na métrica de desvio padrão. Para melhor entendimento dos resultados foi realizada uma análise resumida de cada implementação separadamente a seguir:

**BANANA PI M3 + UBUNTU SERVER:** Neste teste essa implementação obteve um dos piores resultados quando comparados aos resultados obtidos por todas as outras implementações, sendo os piores resultados em quase todas as métricas, obteve na métrica no quesito tempo médio de *upload* de 640,67 segundos e o valor máximo de tempo obtido neste serviço foi de 2.496,00 segundos nas 30 repetições, isso é um valor exorbitante em relação as outras implementações. Todavia obteve um menor uso dos recursos de processamento e memória durante os testes em relação as outras implementações do Raspberry.

**RASPBERRY PI 3 + RASPBIAN STRETCH LITE:** Neste teste essa implementação surpreendeu, obtendo ótimos resultados, superando até mesmo a implementação do servidor DELL T410. Todavia nota-se que o desvio padrão dos seus resultados apresentaram uma alta discrepância, isso o torna um serviço pouco preciso, obteve nas métricas dos quesitos tempo médio de *upload* 289,40 segundos e *download* 120,77 segundos, foram os melhores resultados entre todas as implementações. Percebe-se também que esta implementação obteve um uso muito elevado do processamento e memória 87,21% e 46,56% respectivamente, e a maior velocidade de *upload* e *download* verificada pelo monitoramento da rede com o sistema Zabbix, chegando a casa dos Mbps.

**RASPBERRY PI 3 + RASPBIAN STRETCH LITE + DOCKER:** Neste teste essa implementação surpreendentemente obteve ótimos resultados no teste de disponibilização do serviço StaaS (*Storage as a Service*), com a média 369,03 segundos de *upload* e 812,60 segundos de média de tempo de conclusão dos testes, esses resultados só perderam para as implementações do Raspberry com raspbian sem virtualização e para o servidor DELL T410. Percebe-se também que a implementação chegou a utilizar um nível alto de processamento no teste 67,65% e com o uso de memória mediano 31,14%.

**RASPBERRY PI 3 + UBUNTU MATE + DOCKER:** Neste teste essa implementação obteve resultados semelhantes quando comparado com a implementação de concorrência direta,



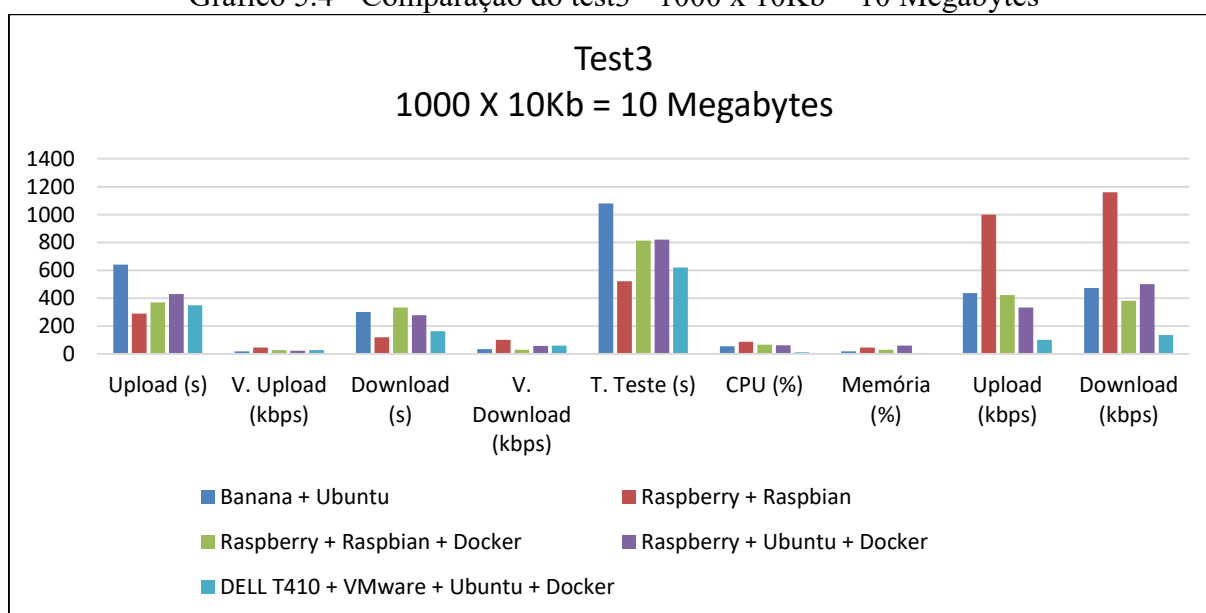
Raspberry Pi com Raspbian Stretch e utilizando a virtualização Docker *Container*, foi o caso das métricas do tempo médio de *upload*, tempo médio de *download* e tempo médio utilizado para a realização dos testes, 430,57 segundos, 278,13 segundos e 821,17 segundos respectivamente. Todavia nota-se que foi a implementação que obteve o mais alto nível do uso de memória do *hardware* 60,82.

### SERVIDOR DELL T410 + UBUNTU SERVER + VMWARE ESXI + DOCKER:

Neste teste essa implementação não conseguiu obter os melhores resultados em todos os aspectos analisados em relação às outras implementações. Nota-se que o tempo médio de 350,63 segundos para *upload* e de 163,93 segundos para *download* foram resultados negativos comparados com a implementação do Raspberry com o raspbian e sem virtualização. Porém, seus resultados foram sem grandes variações nas métricas mínimo e máximo obtidos nos resultados, isso demonstra um serviço muito preciso, o que não acontece com os dispositivos de sistemas embarcados. Outro dado interessante de ser analisado é a métrica referente ao nível máximo do uso do processamento do *hardware* obtendo 12,00% analisado com o sistema de monitoramento Zabbix, um excelente resultado comparado com as outras implementações.

O gráfico 5.4 mostra melhor visualmente a comparação entre todas as implementações referentes ao tempo médio de *upload* e *download*, velocidade média de *upload* e *download*, e tempo médio utilizado na execução dos testes obtidos pelo sistema de *benchmark* Smashbox, também apresenta os valores do pico de uso do processamento, memória, velocidade de *upload* e *download*, capturados com o sistema de monitoramento de rede Zabbix.

Gráfico 5.4 - Comparação do test3 - 1000 x 10Kb = 10 Megabytes



Fonte: Própria do autor (2018)

Neste teste ocorreram os maiores resultados na métrica de desvio padrão dos resultados mínimo e máximo obtido pelos equipamentos de sistemas embarcados em todas as implementações, neste sentido foi feita uma análise com ajuda do sistema de monitoramento de rede Zabbix no momento da realização dos testes para saber o que ocorria nos recursos de processamento, memória e rede de cada equipamento e foi identificado que o uso do processador é o fator crítico na disponibilidade do serviço StaaS (*Storage as a Service*), nas implementações dos dispositivos de sistemas embarcados. A figura 5.4 mostra melhor visualmente a comparação do uso do recurso de processamento entre os equipamentos Banana Pi M3, Raspberry 3 B e o DELL Server T410 executando o teste Test3 0/100/10000.

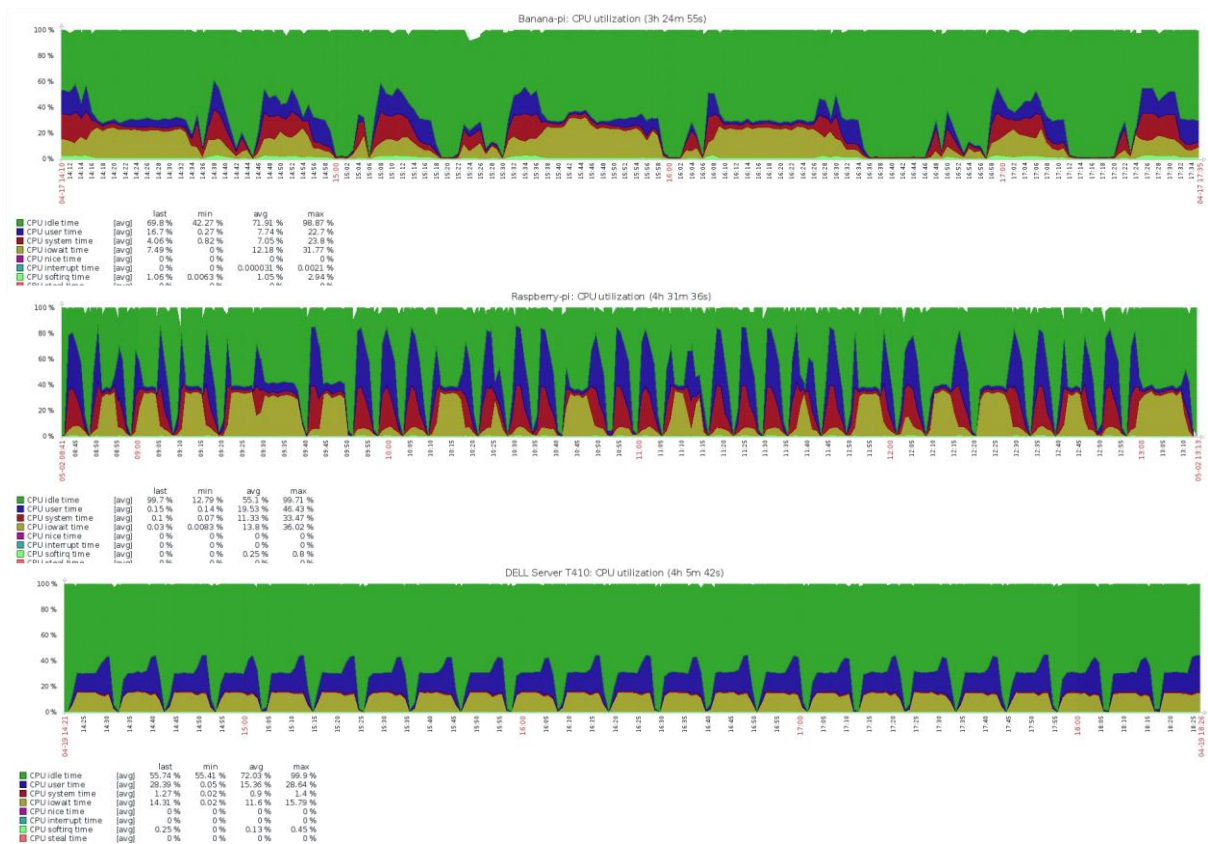


Fig. 5.4 - Utilização da CPU dos Equipamentos  
Fonte: Própria do autor (2018)

Na figura 5.4 percebe-se que os dispositivos Banana Pi M3 e o Raspberry 3 B não exercem uma curva contínua no uso dos seus processos na CPU, principalmente na variável “CPU iowait time” que é o tempo utilizado pelo processador para aguardar a entrada e saída de processos para serem executados, quando isso ocorre o tempo para execução do teste é muito elevado, diferentemente do servidor DELL T410 que apresenta um desenho uniforme em todos os testes, essa anomalia poderá ser analisada melhor em trabalhos futuros por não fazer parte do escopo deste trabalho.

### 5.2.5 Test4 0/1/500000000

O test4 0/1/500000000 consiste no maior volume testado nesse trabalho para a transferência de arquivos em armazenamento em nuvem, esse teste avalia a transferência de um arquivo com o tamanho de 500 Megabytes, é ideal para se analisar como o sistema se comporta com a transferência de um arquivo com volume grande. A tabela 5.5 mostra os resultados coletados do tempo médio utilizado e velocidade de *upload*; tempo médio utilizado e velocidade de *download*; apurar o tempo médio utilizado em cada teste; o desvio padrão, o valor mínimo e máximo obtidos nos resultados com o sistema de *benchmark* Smashbox e em paralelo apresenta as medições do máximo atingido do consumo do processamento, memória RAM e fluxo de dados na rede com o sistema de monitoramento de rede Zabbix em cada implementação testada, o tempo encontra-se em segundos.

Tabela 5.5 - Resultados do Test4 - 1 x 500Mb = 500 Megabytes

BANANA PI M3 + UBUNTU SERVER									
TEST4 0/1/500000000 1 x 500Mb = 500 Megabytes	SMASHBOX				ZABBIX				
	MÉD	DES	MÍN	MAX	CPU %	MEM %	UPL mbps	DOW mbps	
	Tempo de <i>upload</i> em segundos	294,00	24,62	237,00	340,00	40,92	17,00	58,86	70,37
	Velocidade <i>upload</i> em mbps	1,71	0,15	1,47	2,11				
	Tempo <i>download</i> em segundos	36,33	12,69	21,00	81,00				
	Velocidade <i>download</i> em mbps	15,27	4,85	6,17	23,81				
	Tempo do teste em segundos	347,47	31,03	271,00	394,00				
RASPBERRY PI 3 + RASPBIAN STRETCH LITE									
Tempo de <i>upload</i> em segundos	281,83	84,91	174,00	449,00	57,05	33,96	58,89	70,66	
Velocidade <i>upload</i> em mbps	1,92	0,53	1,11	2,87					
Tempo <i>download</i> em segundos	49,40	5,93	44,00	69,00					
Velocidade <i>download</i> em mbps	10,24	1,02	7,25	11,36					
Tempo do teste em segundos	345,43	94,24	226,00	544,00					
RASPBERRY PI 3 + RASPBIAN STRETCH LITE + DOCKER									
Tempo de <i>upload</i> em segundos	394,37	146,64	191,00	886,00	59,41	30,30	54,40	69,88	
Velocidade <i>upload</i> em mbps	1,44	0,53	0,56	2,62					
Tempo <i>download</i> em segundos	55,30	8,26	47,00	81,00					
Velocidade <i>download</i> em mbps	9,20	1,13	6,17	10,64					
Tempo do teste em segundos	476,80	158,12	254,00	1018,00					
RASPBERRY PI 3 + UBUNTU MATE + DOCKER									
Tempo de <i>upload</i> em segundos	346,30	168,15	190,00	850,00	65,00	38,95	58,33	69,70	
Velocidade <i>upload</i> em mbps	1,74	0,67	0,59	2,63					
Tempo <i>download</i> em segundos	63,97	25,32	48,00	161,00					
Velocidade <i>download</i> em mbps	8,48	1,85	3,11	10,40					
Tempo do teste em segundos	437,03	188,62	258,00	961,00					
SERVIDOR DELL T410 + UBUNTU SERVER + VMWARE ESXI + DOCKER									
Tempo de <i>upload</i> em segundos	40,30	3,48	37,00	51,00	12,00	7,08	100,70	135,90	
Velocidade <i>upload</i> em mbps	12,49	0,99	9,80	13,51					
Tempo <i>download</i> em segundos	19,23	3,08	16,00	27,00					
Velocidade <i>download</i> em mbps	26,61	3,99	18,52	31,25					
Tempo do teste em segundos	67,47	5,96	61,00	80,00					

Fonte: Própria do autor (2018)

Nota-se que nesse teste também todas as implementações obtiveram resultados melhores no tempo de *download* em relação ao tempo de *upload* e consequentemente maior velocidade. Também se observa que as implementações nos dispositivos de sistemas embarcados obtiveram um valor muito elevado na métrica de desvio padrão. Para melhor entendimento dos resultados foi realizada uma análise resumida de cada implementação separadamente a seguir:

**BANANA PI M3 + UBUNTU SERVER:** Neste teste essa implementação obteve bons resultados, obteve uma métrica melhor no quesito tempo médio de *download* comparado a implementação do Raspberry com raspbian sem virtualização Docker. Porém seus outros resultados ficaram abaixo comparado ao seu concorrente direto a implementação do Raspberry com raspbian sem virtualização. Todavia obteve um baixo uso de processamento e memória durante os testes em relação as outras implementações do Raspberry.

**RASPBERRY PI 3 + RASPBIAN STRETCH LITE:** Esta implementação continuou obtendo bons resultados, perdendo apenas para a implementação do servidor DELL T410. Todavia nota-se que o desvio padrão dos seus resultados apresentaram uma alta discrepância, obteve na métrica de desvio padrão do tempo médio utilizado para cada teste o resultado de 94,24 segundos, melhor e pior tempo obtido neste parâmetro 226,00 e 544,00 segundos respectivamente, isso o torna um serviço pouco preciso. Percebe-se também que esta implementação obteve um alto uso de processamento e memória 57,05% e 33,96% respectivamente verificada pelo monitoramento da rede com o sistema Zabbix.

**RASPBERRY PI 3 + RASPBIAN STRETCH LITE + DOCKER:** Nesta implementação nota-se que obteve os piores resultados no teste de disponibilização do serviço StaaS (*Storage as a Service*), com a média 394,37 segundos de *upload* e 55,30 segundos de *download*, e uma média de tempo de conclusão dos testes muito elevada 476,80 segundos. Percebe-se também que foi a implementação que chegou a um alto nível de processamento no teste 59,41% sendo menor apenas da implementação do Raspberry com ubuntu e virtualização Docker *container* e com o uso de memória mediano 30,30%.

**RASPBERRY PI 3 + UBUNTU MATE + DOCKER:** Esta implementação obteve bons resultados quando comparado com a implementação de concorrência direta, Raspberry Pi com Raspbian Stretch e utilizando a virtualização Docker *Container*, foi o caso das métricas do tempo médio de *upload* e tempo médio utilizado para a realização dos testes, 346,30 segundos

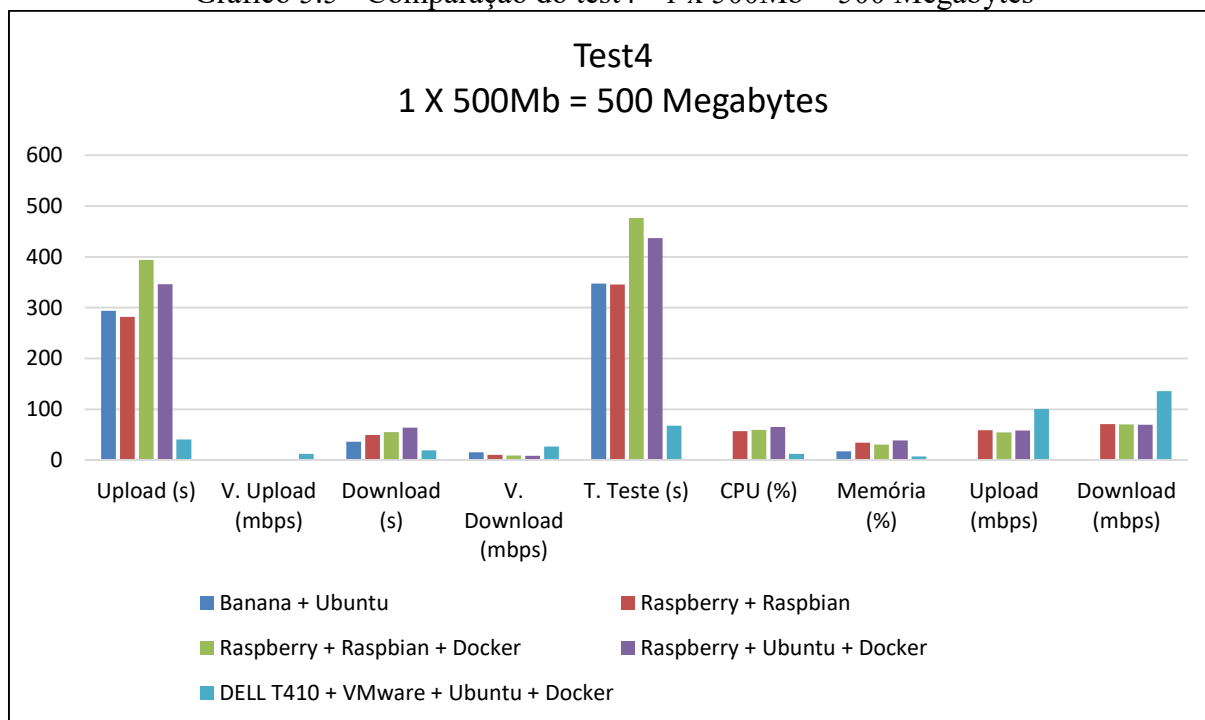
e 437,03 segundos respectivamente. Todavia nota-se que foi a implementação que obteve o mais alto nível do uso da memória do *hardware* 65,00%.

#### SERVIDOR DELL T410 + UBUNTU SERVER + VMWARE ESXI + DOCKER:

Percebe-se que nesse teste essa implementação obteve os melhores resultados em todos os aspectos analisados em relação as outras implementações. Nota-se que o tempo médio de *upload* e *download* são excelentes, sem grandes variações nas métricas mínimo e máximo obtidos nos resultados, isso demonstra um serviço bem preciso, o que não acontece com os dispositivos de sistemas embarcados, e suas métricas relacionadas ao desvio padrão dos resultados obtidos são muito baixas nas 30 repetições do teste. Outros dados interessantes de serem analisados são as métricas referentes a velocidade de *Upload* e *download* obtido na placa de rede pelo sistema de monitoramento de rede Zabbix 100,07Mbps e 135,90Mbps, passando pela primeira vez da barreira dos 100Mbps nos testes, um excelente resultado comparado com as outras implementações. Isso com o pico de uso do processamento e memória muito baixo.

O gráfico 5.5 mostra melhor visualmente a comparação entre todas as implementações referentes ao tempo médio de *upload* e *download*, velocidade média de *upload* e *download*, e tempo médio utilizado na execução dos testes obtidos pelo sistema de *benchmark* Smashbox, também apresenta os valores do pico de uso do processamento, memória, velocidade de *upload* e *download*, capturados com o sistema de monitoramento de rede Zabbix.

Gráfico 5.5 - Comparação do test4 - 1 x 500Mb = 500 Megabytes



Fonte: Própria do autor (2018)

### EXPERIMENTO DA FOG COMPUTING

Neste capítulo é exposto um experimento simulando um sistema de monitoramento com o sistema FogSys para validar o serviço de armazenamento StaaS (*Storage as a Service*), utilizando uma arquitetura *Fog Computing*, instalando a *Cloud Storage Client* na plataforma de dispositivo de sistemas embarcados Raspberry PI para validar o funcionamento da *Fog Computing*.

Foi realizado um experimento em um ambiente controlado de laboratório para simular uma situação real, sendo implementado um sistema de monitoramento de ambiente em uma rede local. Foi desenvolvido um sistema na linguagem Python com o *framework* Django denominado FogSys responsável em simular e capturar dados de dispositivos IoT e enviar para a *Fog Computing* que por sua vez sincroniza com *Cloud Computing* e, por fim, enviar para o dispositivo móvel *Smartphone*. Conforme se mostra na figura 4.2.

Para o experimento utilizaram-se dos mesmos equipamentos para realização das análises dos testes de desempenho do serviço de armazenamento StaaS (*Storage as a Service*), um Raspberry PI 3 modelo “B” com virtualização Docker *container* com a instalação do sistema *Cloud Storage OwnCloud Client* e o sistema FogSys, o desktop DELL OptiPlex 5050 com o sistema FogSys utilizando o simulador de dados de dispositivos IoT e o servidor DELL PowerEdge T410 com instalação do *Cloud Storage Server* e um dispositivo móvel *Smartphone*, utilizando Sistema Operacional (SO) Android e executando *Cloud Storage Client*.

#### 6.1 Sistema FogSys

O sistema teve como objetivo principal simular, receber, validar e armazenar os dados de dispositivos IoT para serem transferidos para *Cloud Computing*, funcionando como uma *Fog Computing* para fornecer o serviço de StaaS (*Storage as a Service*).

Consiste em um programa em arquitetura WEB que armazena todos os processos de requisições de armazenamento de dados realizados por dispositivos IoT previamente cadastrados. Uma das principais funcionalidades é o serviço *WEB Service* simulando e armazenando requisições de dados de dispositivos IoT para serem transferidos a *Cloud Computing*.

A aplicação WEB designa de forma geral um sistema de informática projetado para utilização através de um navegador, através da internet ou aplicativos desenvolvidos utilizando tecnologias WEB, HTML, JavaScript e CSS, pode ser executado a partir de um servidor HTTP (*WEB Host*) ou localmente e no dispositivo do usuário. O sistema informatizado FogSys foi implementado em Docker *Container* totalmente funcional, utilizando as tecnologias WEB, PYTHON, DJANGO, HTML, JAVASCRIPT, CSS, SQLITE.

Django é um *framework* gratuito e de código aberto para a criação de aplicações WEB, escrito em Python. É um *framework* WEB, ou seja, é um conjunto de componentes que ajuda a desenvolver sites de forma mais rápida e mais fácil.

Django é um ambiente para desenvolvimento rápido para WEB escrito em Python que utiliza o padrão MTV (*Model, Template e View*). Criado originalmente para gerenciar um site jornalístico, tornou-se um projeto de código aberto e foi publicado sob a licença BSD em 2005. Seu nome é inspirado no músico de jazz Django Reinhart.

Como o django é escrito em Python, que está sobre a licença *Python Software Foundation License*, que é similar a GPL, com exceção de que se podem distribuir os binários da linguagem sem necessitar anexar as fontes.

Quem trabalha com a dupla python e django sabe os incríveis recursos do *framework*. Os que mais se destacam são:

- Mapeador de objeto relacional;
- URL's elegantes;
- Sistema de *templates*;
- Interface de administração automática (django-admin).

O requisito é uma condição cuja exigência deve ser satisfeita. Se a condição é produzir algo, diz-se que o requisito é funcional. Se a condição é caracterizar algo (atributo, propriedade, comportamento, restrição, etc.), diz-se que o requisito é não-funcional. A descrição destes requisitos é uma atividade indutiva e continuada. Descrever requisitos funcionais e requisitos

não-funcionais requer uma abordagem holística, tratando os dois aspectos: primeiro, "Produzir"; segundo, "com Qualidade", os dois princípios aplicáveis à Engenharia de *Software*.

Desta forma realizou-se o levantamento dos requisitos da interface para a validação, a partir dessa técnica obteve-se os requisitos seguintes:

Requisito funcional - um requisito de sistema de *software* que especifica uma função que o sistema ou componente deve ser capaz de realizar. Estes são requisitos de *software* que definem o comportamento do sistema, ou seja, o processo ou transformação que componentes de *software* ou *hardware* efetuam sobre as entradas para gerar as saídas. Esses requisitos capturam as funcionalidades sob o ponto de vista do usuário.

- Permitir autenticação por meio de login;
- Permitir a recuperação de senha;
- Cadastrar, alterar e excluir usuário do sistema;
- Cadastrar, alterar e excluir sensores no sistema;
- Permitir receber requisição HTTP;
- Permitir receber *string hash* e validar o sensor cadastrado;
- Salvar o resultado em um arquivo;
- Criar um arquivo para cada dia com todas as medições diárias de cada sensor;
- Permitir salvar o arquivo em local específico no servidor;
- Simular envio de requisições para armazenamento dos dados de dispositivos IoT.

Requisito não funcional - em engenharia de sistemas de software, um requisito não funcional de *software* é aquele que descreve não o que o sistema fará, mas como ele fará. Assim, por exemplo, têm-se requisitos de desempenho, requisitos da interface externa do sistema, restrições de projeto e atributos da qualidade. A avaliação dos requisitos não funcionais é feita, em parte, por meio de testes, enquanto que outra parte é avaliada de maneira subjetiva.

- O sistema deve ser multiplataforma;
- Deve utilizar o bando de dados SQLite;
- Ter uma interface amigável;
- Ter a sessão encerrada para usuários inativos por mais de 10 minutos;
- Permitir a confiabilidade dos dados a serem gravados;
- Está disponível 24 horas por dia;
- Ser desenvolvido na linguagem PYTHON, DJANGO, HTML e JAVASCRIPT.



## 6.2 Fluxograma das ações do Sistema FogSys

Com uso do sistema Visutin v8.05 foram modelado os fluxogramas das ações do Sistema FogSys, para as suas duas principais funções, *WEB Service* responsável por receber requisições de armazenamento dos dispositivos IoT e *Simulation* responsável pela simulação das requisições.

O Visustin é um programa de fluxograma automatizado para desenvolvedores de *software* e redatores de documentos. Economizando esforços na documentação com visualização automática de código, reverte a engenharia do seu código-fonte para fluxogramas ou diagramas de atividades da UML. Visustin lê as instruções *if* e *else*, faz um loop e salta e constrói um diagrama totalmente automatizado. Nenhum desenho manual é necessário e suporta uma vasta quantidade de linguagem de programação.

Visustin é ideal para documentação de sistemas informatizados, adicionando fluxogramas à documentação do projeto, diagramas claros informando instantaneamente ao leitor o que até mesmo o código mais complexo faz.

O uso de fluxogramas permite:

- Analisar algoritmos - corrigi erros;
- Seguir a lógica - evita erros;
- Comparar funções - localiza erros;
- Reestruturar código inválido - adiciona qualidade;
- Documentar o sistema - diminui esforço.

Visustin faz revisão de código, verifique a lógica do programa com fluxogramas e fica mais fácil de encontrar bugs antes dos usuários.

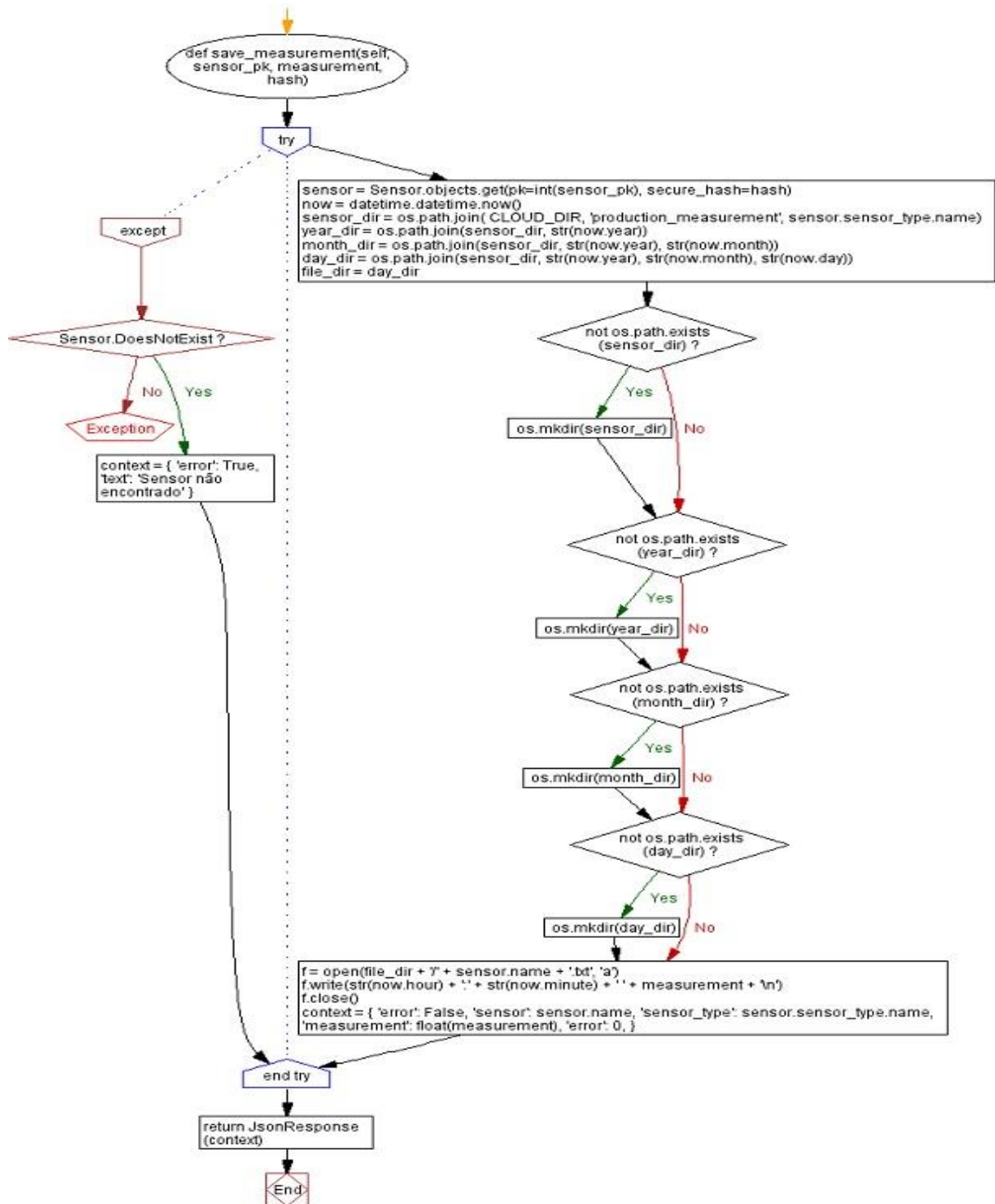
De uma maneira mais específica, o fluxograma da figura 6.1 expõe as ações executadas pelo sistema FogSys a função de *WEB Service*, responsável por receber as requisições de armazenamento dos dispositivos IoT através da URL pelo método GET. Os elementos do fluxograma representam funções ou conjuntos de funções, e de forma mais detalhada é apresentada a seguir:

- Inicialmente é declarada a função `save_measurement()` que recebe através da URL os parâmetros `sensor_pk` que corresponde ao ID do sensor, *measurement* que

corresponde a medição do sensor e hash que corresponde ao número *hash* de cada sensor cadastrado;

- Em seguida entra na função TRY, que verifica as informações enviadas, checando com as disponíveis na base de dados;

Fig. 6.1 - Fluxograma WEB Service FogSys



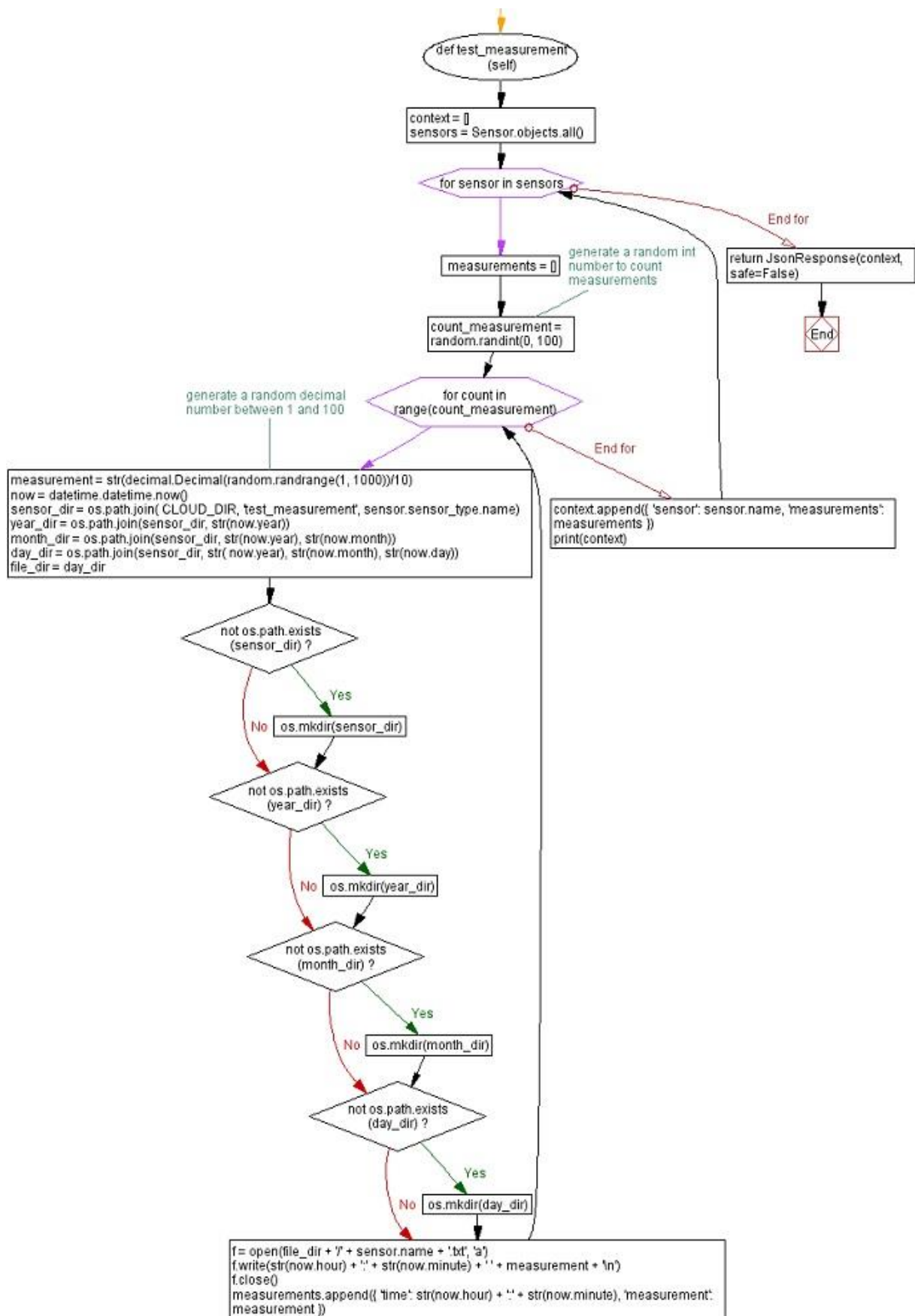
Fonte: Própria do autor (2018)

- Logo após, são criadas as variáveis que formaram o caminho da gravação dos dados no arquivo tipo nome\_sensor.txt, sendo sensor\_dir, year\_dir, month\_dir, day\_dir e file\_dir, os resultados serão gravados no diretório tipo árvore, sendo na ordem tipo do sensor, ano da medição, mês da medição, dia da medição e por fim o arquivo com nome do sensor e suas medições;
- Posteriormente se qualquer diretório não existir o sistema o cria com a função `os.mkdir`;
- Em seguida abre o arquivo com a função `open`, se não existir cria escreve a hora da medição e o valor medido e fechando com a função `close`;
- Logo após verifica se ocorreu algum erro no processo;
- Se os dados informados não atenderem aos requisitos da base de dados entra na função `except` que informa que o sensor não foi encontrado;
- Por fim, retorna à função `JsonResponse` com a variável contexto e é o fim da função.

De uma maneira mais específica, o fluxograma da figura 6.2 expõe as ações executadas pelo sistema FogSys a função de *Simulation*, responsável por simular o envio das requisições de armazenamento dos dispositivos IoT através da URL pelo método *GET*. Os elementos do fluxograma representam funções ou conjuntos de funções, a segunda parte do sistema *Simulation* do FogSys funciona de forma semelhante com a primeira *WEB Service*, divergindo em alguns pontos que são abordados a seguir:

- Inicialmente é declarada a função `test_measurement()` que não recebe nenhum parâmetro;
- Em seguida são criadas as variáveis contexto e instanciado o objeto `sensors` da classe `sensor_objects_all()`;
- Logo após, é gerado um laço para gerar um valor randômico entre 0 e 100 da quantidade de medições para cada sensor;
- Posteriormente é criado outro laço para gerar um valor randômico para cada medição dos sensores entre 1 e 100, esse podendo ser um valor decimal;
- Por fim, o fluxo das ações do sistema FogSys segue semelhante ao da primeira parte do programa *WEB Service*, fornecendo funções para disponibilizar o serviço WEB.

Fig. 6.2 - Fluxograma Simulation FogSys



Fonte: Própria do autor (2018)

### 6.3 Ambiente de desenvolvimento

O sistema FogSys basicamente está dividido em duas partes, a primeira que é o *WEB Service* responsável em receber e gravar em um arquivo e local específico os dados das requisições de armazenamento dos dispositivos IoT para serem transferidos a *Cloud Computing* através do sistema *Owncloud Server e Client*, a segunda parte é a *simulation* responsável pela simulação dos pedidos de requisições de armazenamento dos dispositivos IoT.

O FogSys foi desenvolvido na linguagem Python na versão 3.6 e utilizando o *framework* Django na versão 2.0.3, esse conjunto de ambiente para desenvolvimento foi escolhido por ser uma linguagem e *framework* bastante leve, não necessitando de muito recursos de *hardware* para implementação, sendo ideal para ser executado em plataforma de sistemas embarcados caso do Raspberry PI 3 Modelo “B”.

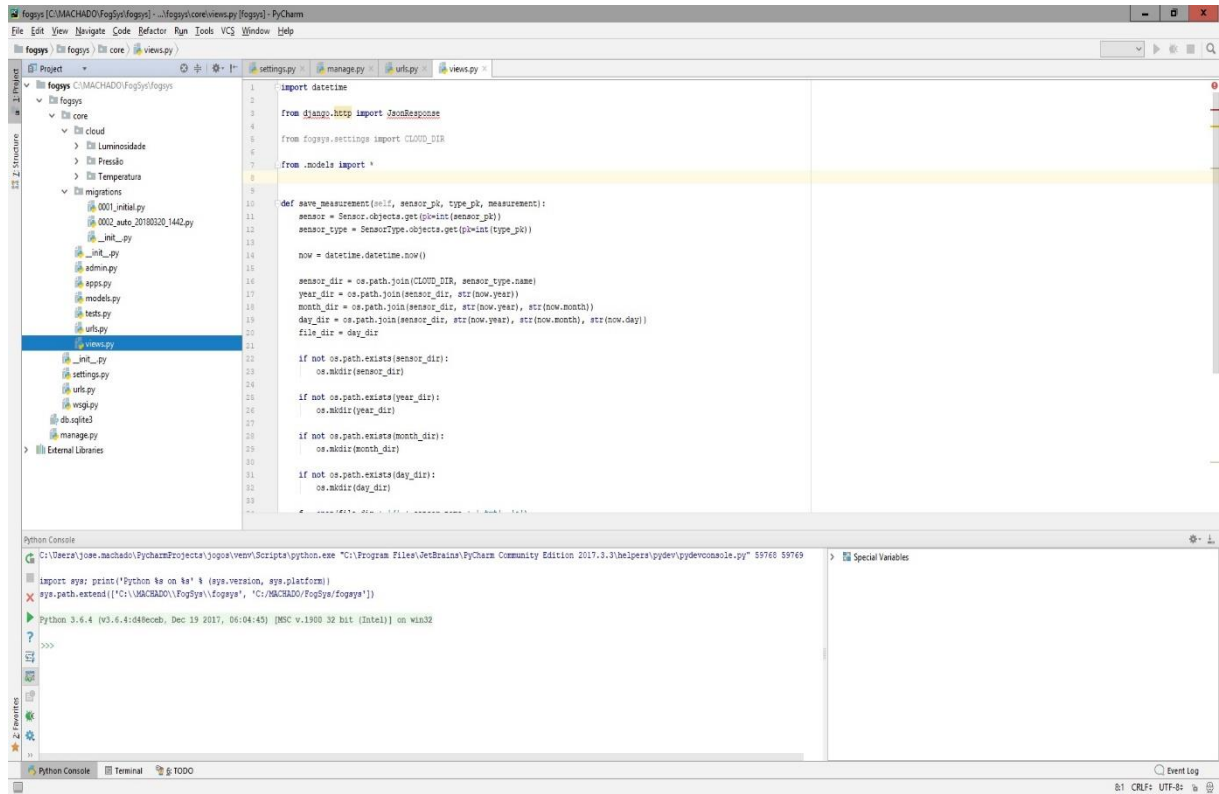
Para o desenvolvimento foi utilizado o IDE PyCharm na versão 2017.3.3. O PyCharm é um IDE (ambiente de desenvolvimento integrado) utilizado para programação em Python, que possui diversos recursos úteis e que facilitam diversas tarefas de desenvolvimento de *software*, principalmente quando comparado com IDE padrão do Python, o IDLE. O PyCharm é desenvolvido pela empresa tcheca JetBrains, sendo escrito em Java e Python, e está disponível para vários sistemas operacionais, como Windows, Linux e OS X. Existem várias licenças disponíveis para o uso do IDE: Professional Edition, que é paga (porém gratuita para alguns casos específicos), e uma versão gratuita e *open source*, a Community Edition, distribuída sob licença Apache.

Entre os principais recursos presentes no PyCharm destaca-se os seguintes:

- Debugs gráfico;
- Unidade de testes integrada;
- Integração com sistemas de controle de versão, como Git, Mercurial e *Subversion*;
- Análise de código;
- *Code Completion*;
- Sintaxe e Erros destacados;
- Refatoração;
- Suporte a desenvolvimento com Django.

A figura 6.3 mostra visualmente o ambiente de desenvolvimento em linguagem Python IDE (*Integrated Development Environment*) PyCharm na versão 2017.3.3 utilizado no projeto de desenvolvimento do sistema FogSys.

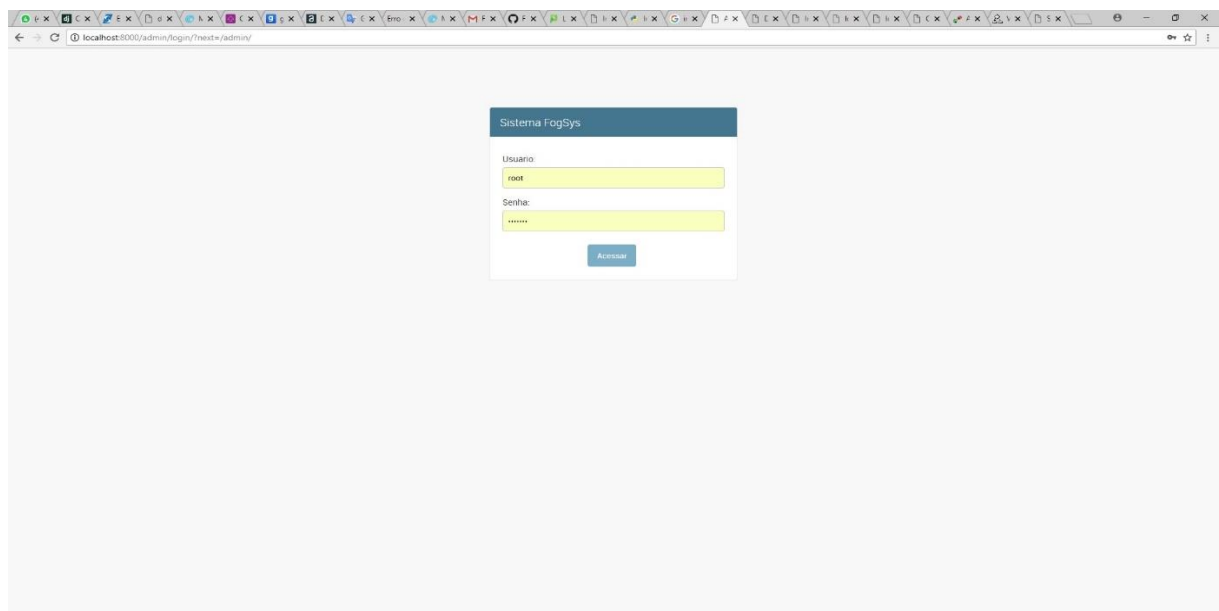
Fig. 6.3 - IDE PyCharm 2017.3.3



Fonte: Própria do autor (2018)

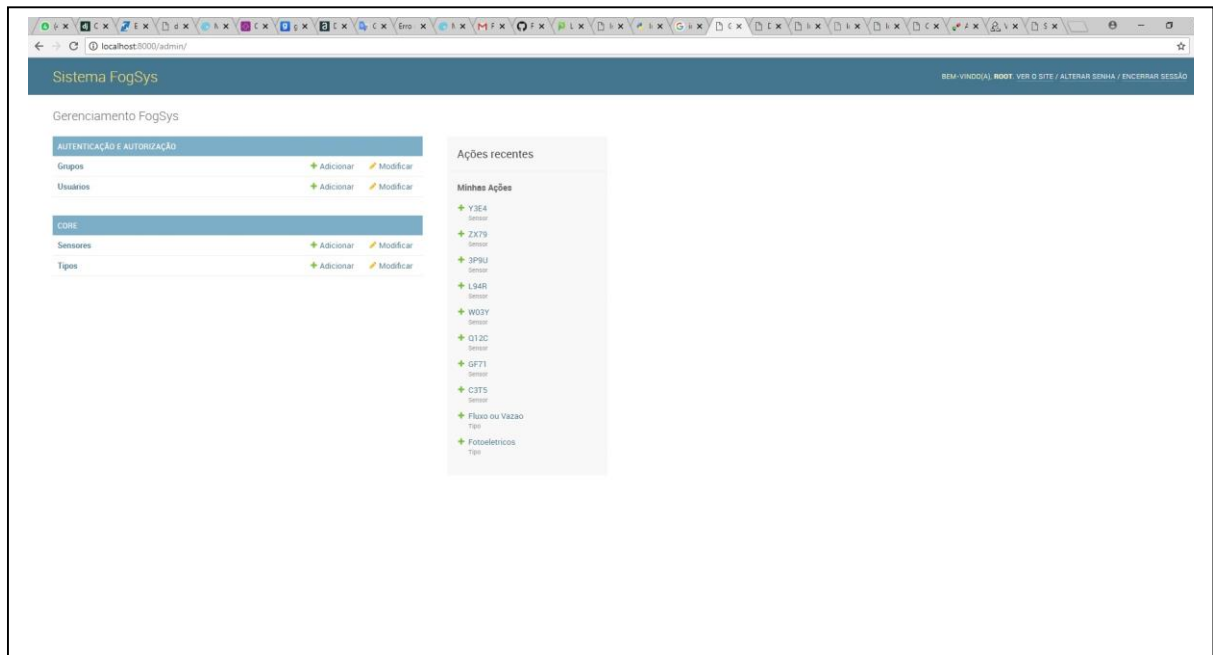
As figuras 6.4, 6.5 e 6.6 mostram visualmente as telas de interface do Sistema FogSys, tela de interface de Login, tela de interface de Gerenciamento do Sistema e tela de interface de cadastro de sensores, respectivamente.

Fig. 6.4 - Interface de Login do FogSys



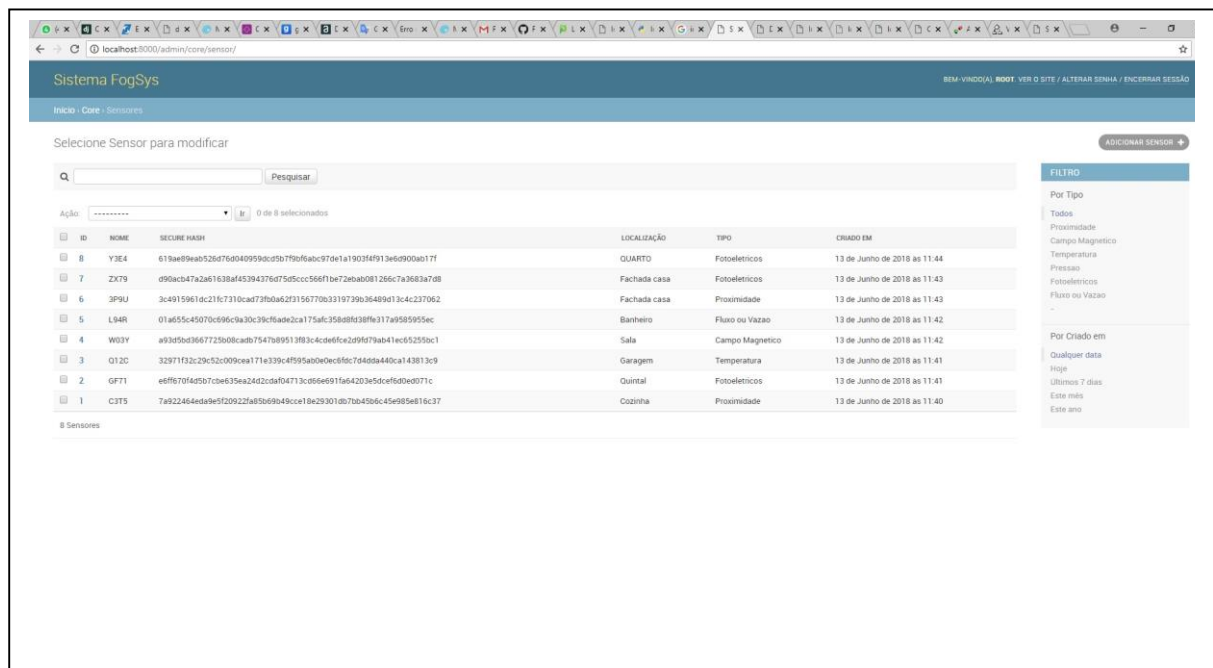
Fonte: Própria do autor (2018)

Fig. 6.5 - Interface de Gerenciamento do FogSys



Fonte: Própria do autor (2018)

Fig. 6.6 - Interface Gerenciamento dos Sensores no FogSys



Fonte: Própria do autor (2018)

Observa-se que todos os sensores cadastrados têm um campo chamado SECURE HASH de 64 caracteres, esse campo é gerado automaticamente pela função `create_hash` na hora do cadastro do sensor com a criação de mensagem de *hashing* que usa o SHA256, concatenado o nome do sensor, a localização, data e hora da sua criação isso gera uma chave única que deverá

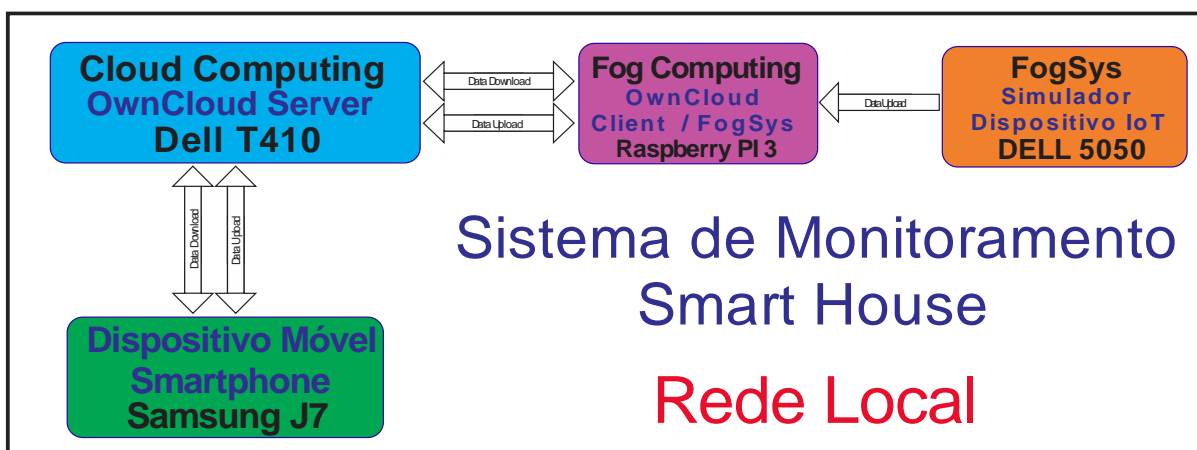
ser enviado pelo sensor através da URL informando sua identidade para autenticação e autorização.

## 6.4 Experimento da Fog Computing com o Sistema FogSys

Foi realizado um experimento em um ambiente controlado de laboratório para simular uma situação real, sendo implementado um sistema de monitoramento de ambiente *Smart House* em uma rede local.

Para o experimento foram utilizados os mesmos equipamentos para realização das análises dos testes de desempenho do serviço de armazenamento StaaS (*Storage as a Service*), um Raspberry PI 3 modelo “B” com virtualização Docker *container* com a instalação do sistema *Cloud Storage OwnCloud Client*, o sistema FogSys, o servidor DELL PowerEdge T410 com instalação do *Cloud Storage Server*, um dispositivo móvel *Smart Phone* Samsung J7, utilizando Sistema Operacional (SO) Android e o dispositivo desktop DELL OptiPlex 5050 foi alocado com o suporte ao sistema FogSys com o módulo de simulador para dispositivos IoT, para ser o gerador de carga e simular um ambiente real, sendo que todos esses sistemas foram virtualizados com Docker *Container*, exceto o Smartphone Samsung Galaxy J7 que foi instalado o sistema de sincronização *Cloud Storage Client* para o OwnCloud denominado cottonCloud versão 2.5.1. A figura 6.7 ilustra o fluxo dos dados na realização do experimento.

Fig. 6.7 - Fluxo dos dados no experimento



Fonte: Própria do autor (2018)

A figura 6.8 apresenta o cenário real da realização do experimento que simula um ambiente de *Fog Computing*.



Fig. 6.8 - Cenário do experimento usando FogSys



Fonte: Própria do autor (2018)

Na imagem percebe-se todos os equipamentos utilizados no experimento, exceto o servidor DELL T410 que estava localizado na sala ao lado, o *Acess Point* RUCKUS ZoneFlex 7363 serviu para fazer a comunicação entre o *Smartphone* Samsung J7 e os outros equipamentos na rede, o *Switch* HP A5120 24 portas 10/100/1000 Base-T interligou todos os equipamentos.

No experimento utilizou-se o conjunto de *hardware* e *software* com RASPBERRY PI 3 + UBUNTU MATE + DOCKER CONTAINER, para fornecer o serviço de armazenamento StaaS (*Storage as a Service*), pelo motivo de ter obtido bons resultados de performance quando comparado com a implementação de concorrência direta RASPBERRY PI 3 + RASPBIAN STRETCH LITE + DOCKER CONTAINER e a virtualização ser uma das características da *Fog Computing*.

Primeiramente foi implementado um ambiente para hospedar o sistema FogSys utilizando um *Docker Container*, com o sistema operacional Linux Ubuntu na versão 18.04 e instalaram-se todas as dependências para o seu funcionamento como: Linguagem Python na versão 3.6, o *framework* Django na versão 2.0.3, o sistema OwnCloud *Client*, o CRON sistema de

agendamento de tarefa do Linux, etc. Em seguida este *container* foi disponibilizado na condição pública no Docker Hub para as arquiteturas x86, x64 e ARM, nos endereços <https://hub.docker.com/r/jsmac/fogsys/> e <https://hub.docker.com/r/jsmac/fogsys-arm/> respectivamente, com as informações para poder executar o sistema FogSys no *container*, podendo ser utilizados pela comunidade científica para futuras pesquisas acadêmicas.

Na configuração do sistema de sincronização dos dados foi criada uma pasta chamada *cloud* no sistema Owncloud *Server* para sincronização dos dados recebidos e agendada uma tarefa no CRON do sistema operacional Linux para sincronizar os dados a cada minuto com o servidor do sistema OwnCloud *Server*, praticamente em tempo real, foi editado o arquivo CRONTAB que se encontra no diretório */etc/* do sistema Linux com a seguinte linha:

```
* * * * * root owncloudcmd -u admin -p admin /fogsys/FogSys/fogsys/fogsys/core/cloud  
http://10.65.1.26/remote.php/webdav/cloud
```

Onde-se,

- Os asteriscos identificam: [minutos] [horas] [dias do mês] [mês] [dias da semana];
- root: [usuário do sistema Linux];
- owncloudcmd: [comando de sincronização do sistema Owncloud *Client*];
- -u: [parâmetro para identificar o usuário do sistema Owncloud *Server*];
- -p: [parâmetro para identificar o *password* do sistema Owncloud *Server*];
- /fogsys/FogSys/fogsys/fogsys/core/cloud: [caminho da pasta a ser sincronizada];
- http://10.65.1.26/remote.php/webdav/cloud: [endereço da pasta no OwnCloud *Server* para sincronização].

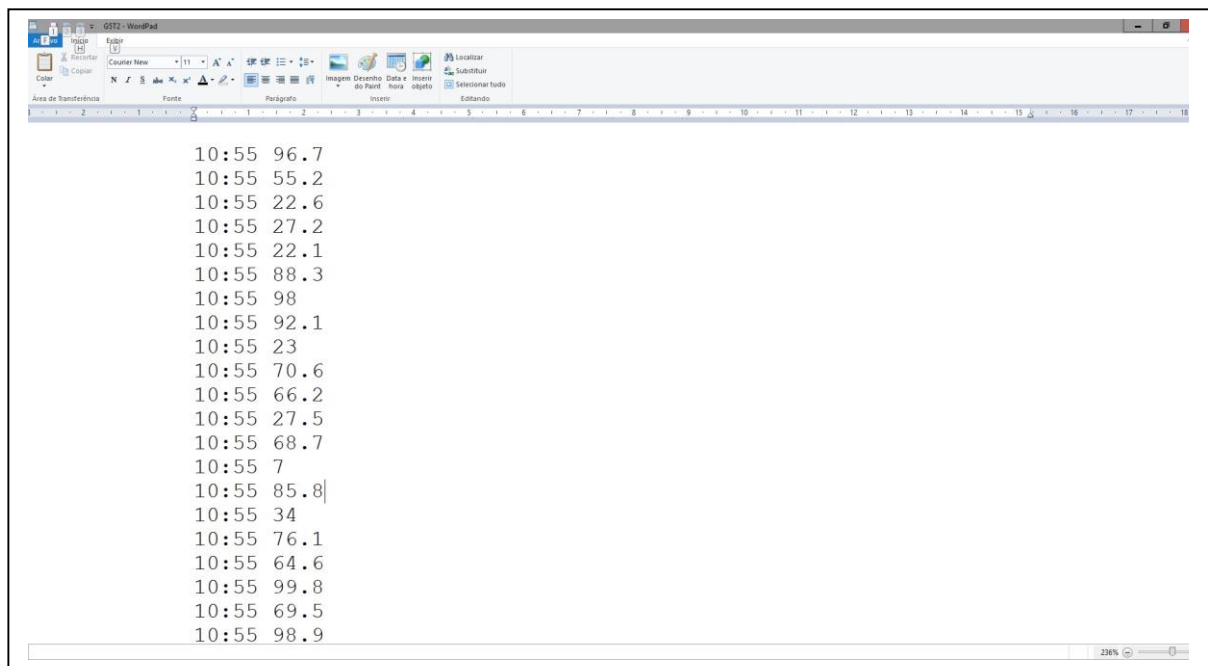
Instruções para utilização do container disponibilizado no Docker Hub no endereço jsmac/fogsys e jsmac/fogsys-arm são as seguintes:

- Execute o contêiner: [docker run -dti --name=fogsys -p 8000:8000 jsmac/fogsys:latest];
- Acesse o contêiner: [docker exec -ti fogsys bash];
- Inicie o servidor do Django: [cd fogsys/Fogsys/fogsys/], [python3.6 manage.py runserver 0.0.0.0:8000];
- Configure o serviço de sincronização criando uma pasta no OwnCloud *Server* denominada *cloud*;
- Edite o arquivo CRONTAB: [nano /etc/crontab];

- Altere o nome de usuário, senha e endereço do Owncloud *Server* na linha abaixo  
[owncloudcmd -u admin -p admin /fogsys/FogSys/fogsys/fogsys/core/cloud  
http://10.65.1.26/remote.php/webdav/cloud];
- Inicie o serviço CRON: [/etc/init.d/cron start]
- Acesse o sistema FogSys para gerenciamento: [http://localhost:8000/admin/];
- Usuário: root senha: admin;
- Exemplo para enviar solicitação via http: [http://localhost:8000/fogsys/6/87.22/  
da109df0e170e6d4b85c4bc28f1535c94d0cd6e28301a5816151f56b8ae2ce06];
- Para execução de teste: [http://localhost:8000/test-measurement/].

A figura 6.9 apresenta um exemplo do conteúdo do arquivo gerado pela simulação de envio dos dados do sensor G5T2 e enviado para *Fog Computing*, observa-se que o nome do arquivo é mesmo nome do sensor no formato txt, onde consta a hora da medição e o valor medido.

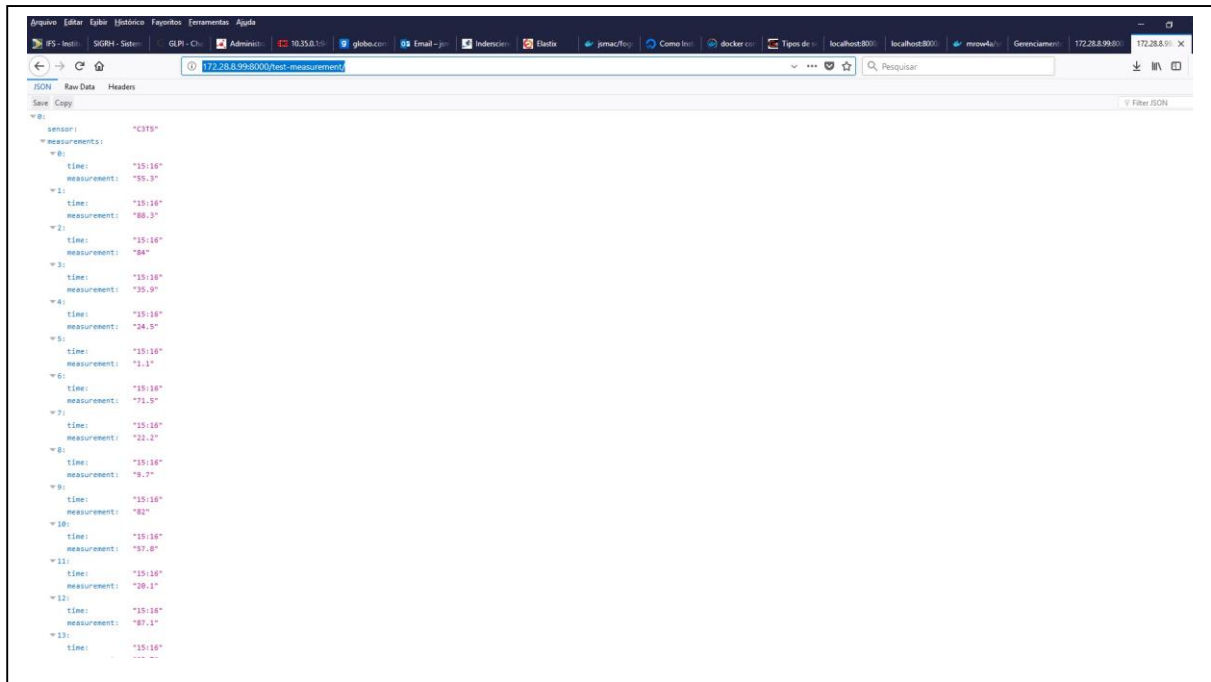
Fig. 6.9 - Arquivo de medição



Fonte: Própria do autor (2018)

A figura 6.10 apresenta a realização do experimento em tempo real da simulação do envio dos dados dos sensores para o sistema FogSys, simulando um ambiente real da *Fog Computing* para fornecer o serviço de armazenamento StaaS (*Storage as a Service*), e o resultado é mostrado no navegador WEB.

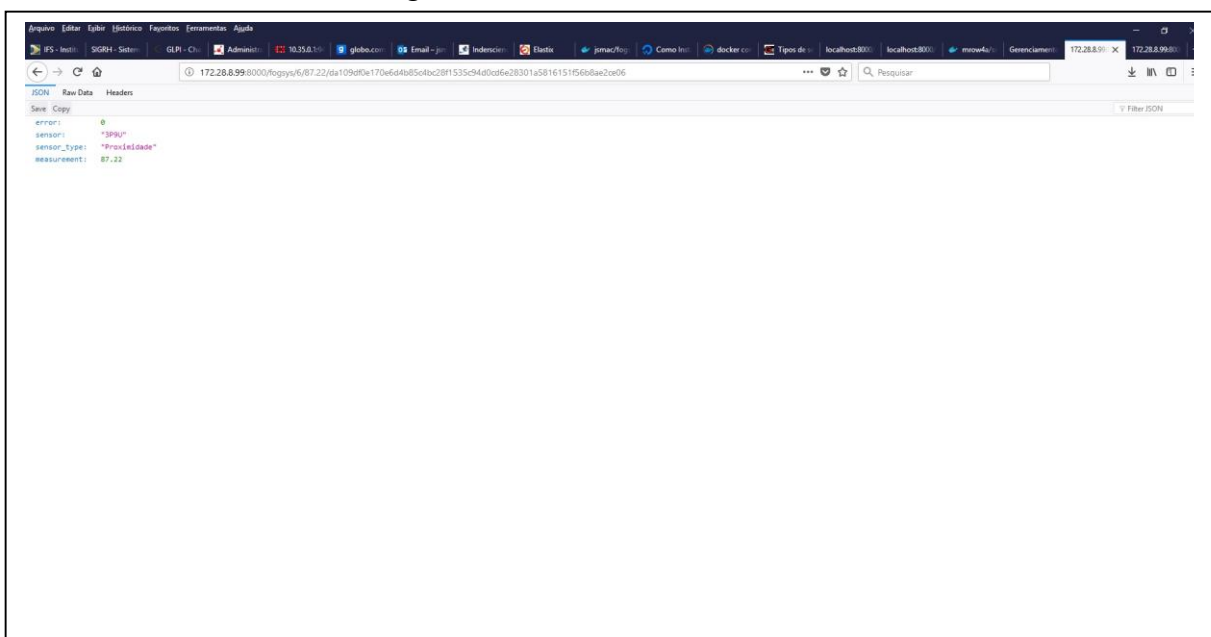
Fig. 6.10 - Teste de simulação



Fonte: Própria do autor (2018)

A figura 6.11 apresenta a realização do experimento em tempo real da medição de produção do envio dos dados dos sensores via protocolo HTTP para o sistema FogSys, simulando um ambiente real da *Fog Computing* para fornecer o serviço de armazenamento SaaS (*Storage as a Service*), e o resultado é mostrado no navegador WEB.

Fig. 6.11 - Teste de envio via HTTP



Fonte: Própria do autor (2018)

Observa-se que o envio dos dados pelos sensores via HTTP segue uma ordem no link de envio, a exemplo:

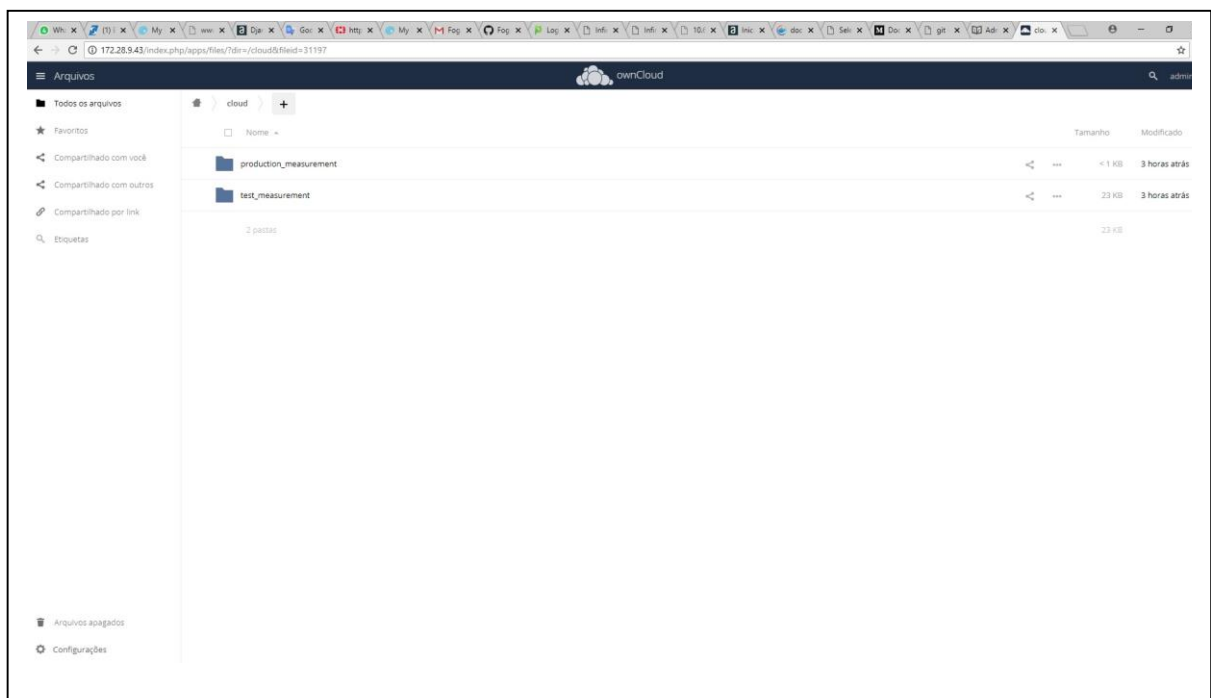
<http://172.28.8.99:8000/fogsys/6/87.22/da109df0e170e6d4b85c4bc28f1535c94d0cd6e28301a5816151f56b8ae2ce06>

Onde-se:

- <http://172.28.8.99:8000/fogsys/>: [endereço do servidor do sistema FogSys];
- /6: [ID do sensor no *database* do sistema FogSys];
- /87.22: [medição do sensor];
- /da109df0e170e6d4b85c4bc28f1535c94d0cd6e28301a5816151f56b8ae2ce06: [número hash que identifica o sensor no *database* do sistema FogSys].

A figura 6.12 apresenta a realização do experimento em tempo real da chegada dos dados na *Cloud Computing* enviados pelo sistema FogSys, simulando um ambiente real da *Fog Computing* para fornecer o serviço de armazenamento StaaS (*Storage as a Service*), e o resultado é mostrado no sistema de armazenamento em nuvem OwnCloud Server.

Fig. 6.12 - Dados no OwnCloud Server

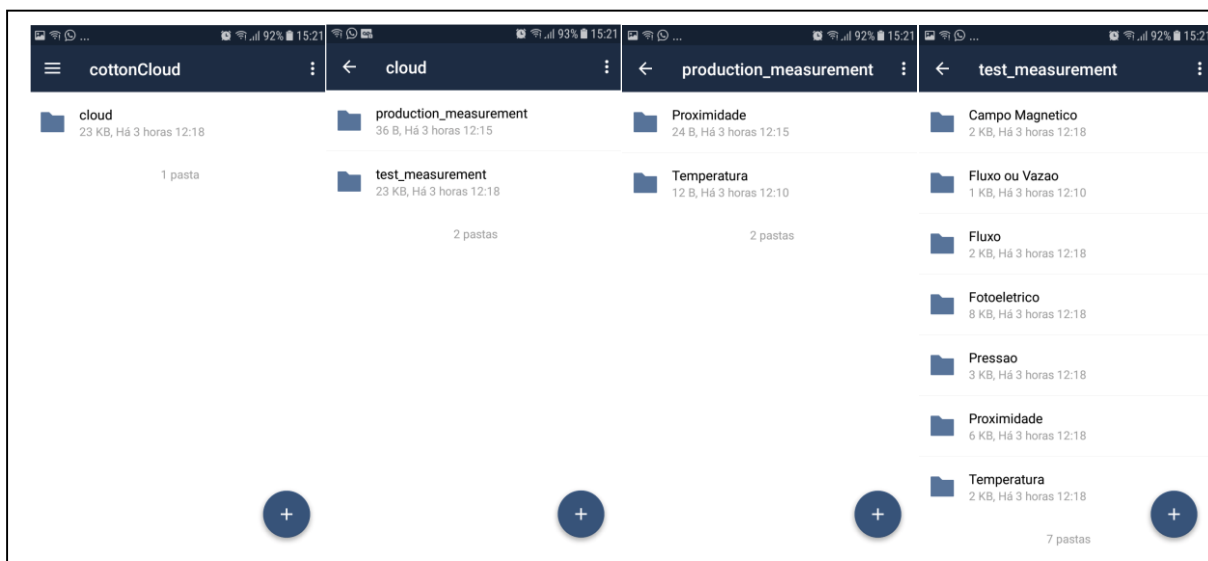


Fonte: Própria do autor (2018)

Observa-se que o sistema cria duas pastas distintas, uma para a medição de produção pelo envio de dados via protocolo HTTP denominada *production\_measurement* e outra para a medição via teste de simulação denominada *test\_measurement*.

A figura 6.13 demonstra a realização do experimento em tempo real da chegada dos dados no *Smartphone* Samsung Galaxy J7 enviados pelo sistema de *Cloud Computing* OwnCloud *Server*, simulando um ambiente real da *Fog Computing* para fornecer o serviço de armazenamento StaaS (*Storage as a Service*), e o resultado é mostrado no *Smartphone* Samsung Galaxy J7 utilizando o sistema de sincronização de dados cottonCloud.

Fig. 6.13 - Dados no Smartphone



Fonte: Própria do autor (2018)

Observa-se que a figura 6.13 demonstra quatro telas diferentes dos dados sincronizados para o *Smartphone* Samsung Galaxy J7, dentro das pastas *production\_measurement* e *test\_measurement* o sistema cria subpastas distintas para cada tipo de sensores cadastrados no sistema FogSys, dentro dessas pastas são criadas subpastas identificando o ano, mês e dia da medição dos sensores e por fim o sistema grava um arquivo com o nome do sensor contendo a hora e o valor das suas medições no formato txt.

Definir o nível aceitável do fornecimento do serviço de armazenamento StaaS (*Storage as a Service*), em uma emergente tecnologia em que se trata o ambiente *Fog Computing* é uma tarefa complexa e subjetiva, dependente da classificação da aplicação.

Diferentes aplicações de computação em *Fog* foram sugeridas na literatura. Segundo OSANAIYE *et al.* (2017), as categorias das aplicações de computação em *Fog* são divididas em 3: (i) Aplicações em tempo real; (ii) Aplicações quase em tempo real; (iii) Aplicações introduzidas em redes.

- (i) As aplicações em tempo real são aplicações de baixa latência, que funcionam dentro de um período de tempo pré-definido, sendo classificada pelo usuário como imediata ou urgente.
- (ii) Quase em tempo real, por outro lado, são aplicações que estão sujeitas a atraso de tempo introduzido pelo processamento de dados ou transmissão de rede, entre o momento em que ocorre um evento e o uso dos dados para processamento.
- (iii) A computação em *Fog* também pode ser introduzida em uma rede (para aplicativos que não necessite de processamento e transmissão em tempo real), para reduzir a quantidade de tráfego no núcleo de processamento.

Neste sentido, nota-se neste trabalho que na avaliação e no experimento da implementação de uma *Fog Computing* para fornecer o serviço de armazenamento StaaS (*Storage as a Service*), utilizando o sistema FogSys com o dispositivo de sistemas embarcados Raspberry 3 Pi, foi realizado com êxito e podendo satisfazer as expectativas para as aplicações classificadas em aplicações quase em tempo real e introduzida na rede, não sendo adequada para as aplicações classificadas como aplicações em tempo real, devido as implementações nos dispositivos de sistemas embarcados obterem valores muito elevados nas métricas de desvio padrão, tornando o serviço pouco preciso.

Porém, sabemos que o sistema FogSys é um protótipo e serão necessárias mais pesquisas acadêmicas para aglutinar novas funções que se adéquem aos futuros dispositivos de sensores do mundo IoT.

### CONCLUSÃO

A *Fog Computing* poderá distribuir serviços avançados de computação, armazenamento, rede e gerenciamento mais próximos dos usuários finais ao longo da borda da rede, entre a *Cloud*, IoT e dispositivos móveis, formando assim uma plataforma distribuída e virtualizada.

Problemas de latência, armazenamento e poder de processamento desses dispositivos fizeram surgir a *Fog Computing*. Assim, a computação em *Fog* rapidamente atraiu muita atenção da indústria e da academia.

Esta dissertação de mestrado apresenta o conceito da *Fog Computing*, seus desafios, sua contextualização teórica, os trabalhos correlatos junto com a revisão sistemática e realiza a implementação e análise de uma *Fog Computing*, para fornecer StaaS (*Storage as a Service*), a dispositivos IoT utilizando plataformas de sistemas embarcados e compara seus resultados com os obtidos por um servidor de alto desempenho. Foram realizados testes com vários tamanhos de arquivos (variando de 1 byte até 500 megabytes).

Foram realizados cinco (5) implementações de diferentes combinações de *softwares* e *hardwares*, e analisa seus resultados com a finalidade de encontrar a melhor opção para disponibilizar o serviço de armazenamento StaaS (*Storage as a Service*) em um ambiente *Fog Computing*.

Foi desenvolvido o Sistema FogSys com o objetivo principal de simular, receber, validar e armazenar os dados de dispositivos IoT para serem transferidos para *Cloud Computing*, funcionando como uma *Fog Computing* para fornecer o serviço de StaaS (*Storage as a Service*).

Realiza um experimento da implementação de uma *Fog Computing* para fornecer o serviço de armazenamento StaaS (*Storage as a Service*), utilizando o sistema FogSys com o dispositivo de sistemas embarcados Raspberry 3 Pi, sendo realizado com êxito, porém sabemos que o sistema FogSys é um protótipo e são necessárias mais pesquisas acadêmicas para aglutinar novas funções que se adequem aos futuros dispositivos de sensores do mundo IoT.



Os resultados satisfizeram as expectativas, na avaliação do teste Test3 0/1000/10000 (transferência de 1000 arquivos de 10 Kilobytes) a implementação RASPBERRY PI 3 + RASPBIAN STRETCH LITE surpreendeu, obtendo ótimos resultados, superando até mesmo a implementação do servidor DELL T410, obteve nas métricas dos quesitos tempo médio de *upload* 289,40 segundos e *download* 120,77 segundos contra *upload* de 350,63 segundos e *download* 163,93 segundos do servidor DELL T410, redução 17,46% e 26,33% respectivamente, foram os melhores resultados entre todas as implementações neste teste. Todavia notou-se que o desvio padrão dos seus resultados apresentou uma alta discrepância, isso o torna um serviço pouco preciso.

Neste sentido, nota-se neste trabalho que na avaliação e no experimento da implementação de uma *Fog Computing* para fornecer o serviço de armazenamento StaaS (*Storage as a Service*), utilizando o sistema FogSys com o dispositivo de sistemas embarcados Raspberry 3 Pi, foi realizado com êxito e podendo satisfazer as expectativas para as aplicações classificadas em aplicações quase em tempo real e introduzida na rede, não sendo adequada para as aplicações classificadas como aplicações em tempo real, devido as implementações nos dispositivos de sistemas embarcados obterem valores muito elevados nas métricas de desvio padrão, tornando o serviço pouco preciso.

Portanto percebe-se que a implementação desse serviço em dispositivos de sistemas embarcados pode ser uma boa alternativa para reduzir um desses problemas, no caso, o armazenamento de dados, que atinge hoje os dispositivos IoT, servindo como *Fog Computing* para armazenar dados desses dispositivos e enviar à *Cloud Computing*, com segurança e eficiência no transporte dos dados entre as redes, e sendo implementado num dispositivo de plataforma embarcada de baixo custo, em vez de usar potentes e caros servidores para exercer essa função.

## 7.1 Artigos publicados

Nesta dissertação foram publicados os seguintes artigos:

- Uma Revisão da Fog Computing e Seus Desafios de Pesquisas - **Journal on Advances in Theoretical and Applied Informatics** – 2017;
- A Survey on Fog Computing and Its Research Challenges - **International Journal of Grid and Utility Computing** – 2018.

Nos apêndices A e B apresentam as versões completas desses artigos.

## 7.2 Trabalhos futuros

Com a finalidade e o desejo em dar continuidade ao projeto de implementação de *Fog Computing* utilizando dispositivos de sistemas embarcados, acredita-se que esse trabalho abriu vários cenários para novos trabalhos futuros pela comunidade acadêmica científica.

O desempenho das cinco implementações abordados neste trabalho foram avaliados com a finalidade de encontrar a melhor combinação de *software* e *hardware* para disponibilizar o serviço de armazenamento StaaS (*Storage as a Service*). Não obstante, novos dispositivos embarcados, assim como novas tecnologias, surgirão. Dessa forma, amplia-se a possibilidade de estender este trabalho, sendo:

- Analisar o serviço de armazenamento StaaS (*Storage as a Service*), com diferentes plataformas de dispositivos de sistemas embarcados não utilizados neste trabalho e utilizando o *benchmark* Smashbox;
- Analisar o serviço de armazenamento StaaS (*Storage as a Service*), com diferentes plataformas de serviço de armazenamento em nuvem, caso do sistema Seafile utilizando o *benchmark* Smashbox;
- Analisar mais profundamente o *hardware* (como processamento, memória e rede) dos dispositivos de sistemas embarcados para descobrir o fato dos seus resultados nas avaliações do fornecimento do serviço de armazenamento StaaS (*Storage as a Service*), obterem uma enorme discrepância no valor do desvio padrão;
- Realizar a mesma avaliação e experimento utilizando *cluster* com dispositivos de sistemas embarcados, usando a virtualização DOCKER *Container* e comparar os resultados obtidos com deste trabalho;
- Implementar a *Fog Computing* com o sistema FogSys utilizando sensores reais;
- Desenvolver novas funções para o sistema FogSys que se adequem aos futuros dispositivos de sensores do mundo IoT.

Além das técnicas de medições apresentadas é possível acoplar novas técnicas e assim substituí-las ou até mesmo aperfeiçoá-las. Realizar testes com técnicas diferentes é importante para definir a técnica que possui melhor resultado.

## REFERÊNCIAS

---

AL-DOGHMAN, Firas et al. A review on Fog Computing technology. In: **Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on**. IEEE, 2016. p. 001525-001530.

ALRAWAIS, Arwa et al. Fog Computing for the Internet of Things: Security and Privacy Issues. **IEEE Internet Computing**, v. 21, n. 2, p. 34-42, 2017.

ANTONI, Marco; VIVIAN, Gláucio Ricardo; PREUSS, Evandro. Implementação de uma nuvem de armazenamento privada usando Owncloud e Raspberry PI. **Anais do EATI - Encontro Anual de Tecnologia da Informação e Semana Acadêmica de Tecnologia da Informação**, 2015, Ano 5 n. 1, pp. 55-62.

BABU, Shaik Masthan; LAKSHMI, A. Jaya; RAO, B. Thirumala. A study on cloud based internet of things: Cloudiot. In: **Communication Technologies (GCCT), 2015 Global Conference on**. IEEE, 2015. p. 60-65.

BOTTA, Alessio et al. Integration of cloud computing and internet of things: a survey. **Future Generation Computer Systems**, v. 56, p. 684-700, 2016.

BPI-M3 Octa-core Development Board. Disponível em: <http://www.banana-pi.org/m3.html>. Acesso outubro 2017.

CHEN, Songqing; ZHANG, Tao; SHI, Weisong. Fog Computing. **IEEE Internet Computing**, v. 21, n. 2, p. 4-6, 2017.

CHIANG, Mung et al. Clarifying Fog Computing and Networking: 10 Questions and Answers. **IEEE Communications Magazine**, v. 55, n. 4, p. 18-20, 2017.

CRACIUNESCU, Razvan et al. Implementation of Fog computing for reliable E-health applications. In: **Signals, Systems and Computers, 2015 49th Asilomar Conference on**. IEEE, 2015. p. 459-463.

DALLE VACCHE, Andrea; LEE, Stefano Kewan. **Zabbix network monitoring essentials**. Packt Publishing Ltd, 2015.

Detalhes do Servidor DELL PowerEdge T410 em: <http://www1.la.dell.com/br/pt/corp/Servidores/server-poweredge-t410>. Acesso maio 2018.

DÍAZ, Manuel; MARTÍN, Cristian; RUBIO, Bartolomé. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. **Journal of Network and Computer Applications**, v. 67, p. 99-117, 2016.

FAN, Chih-Tien et al. Web Resource Cacheable Edge Device in Fog Computing. In: **Parallel and Distributed Computing (ISPDC), 2016 15th International Symposium on**. IEEE, 2016. p. 432-439.

GENG, Xiaolin et al. An Novel Architecture and Inter-process Communication Scheme to Adapt Chromium Based on Docker Container. **Procedia Computer Science**, v. 107, p. 691-696, 2017.

HAJIBABA, Majid; GORGIN, Saeid. A review on modern distributed computing paradigms: Cloud computing, jungle computing and fog computing. **CIT. Journal of Computing and Information Technology**, v. 22, n. 2, p. 69-84, 2014.

HAO, Zijiang et al. Challenges and Software Architecture for Fog Computing. **IEEE Internet Computing**, v. 21, n. 2, p. 44-53, 2017.

HUO, Jiuyuan; QU, Hong; WU, Ling. Design and implementation of private cloud storage platform based on OpenStack. In: **Smart City/SocialCom/SustainCom (SmartCity), 2015 IEEE International Conference on**. IEEE, 2015. p. 1098-1101.

JAIN, Akshay; SINGHAL, Priyank. Fog computing: Driving force behind the emergence of edge computing. In: **System Modeling & Advancement in Research Trends (SMART), International Conference**. IEEE, 2016. p. 294-297.

JAIN, Raj. **The Art of Computer Systems Performance Analysis: techniques for experimental design, measurement, simulation, and modeling**. 1991.

KITCHENHAM, Barbara. Procedures for performing systematic reviews. **Keele, UK, Keele University**, v. 33, n. 2004, p. 1-26, 2004.

KUMAR, Praveen; ZAIDI, Nabeel; CHOUDHURY, Tanupriya. Fog computing: Common security issues and proposed countermeasures. In: **System Modeling & Advancement in Research Trends (SMART), International Conference**. IEEE, 2016. p. 311-315.

LI, Songze; MADDAAH-ALI, Mohammad Ali; AVESTIMEHR, A. Salman. Coding for Distributed Fog Computing. **IEEE Communications Magazine**, v. 55, n. 4, p. 34-40, 2017.

LINTHICUM, David S. Connecting Fog and Cloud Computing. **IEEE Cloud Computing**, v. 4, n. 2, p. 18-20, 2017.

LONGO, Francesco et al. Stack4things: An openstack-based framework for iot. In: **Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on**. IEEE, 2015. p. 204-211.

MACHADO, José dos Santos; MORENO, Edward David; RIBEIRO, Admilson de Ribamar Lima. A Review of Computing Fog and its Research Challenges. **Journal on Advances in Theoretical and Applied Informatics**, [S.l.], v. 3, n. 2, p. 32-39, dec. 2017. ISSN 2447-5033.

MARTINI, Ben; CHOO, Kim-Kwang Raymond. Cloud storage forensics: ownCloud as a case study. **Digital Investigation**, v. 10, n. 4, p. 287-299, 2013.

MCMILLIN, Bruce; ZHANG, Tao. Fog Computing for Smart Living. **Computer**, v. 50, n. 2, p. 5-5, 2017.

MRÓWCZYŃSKI, Piotr et al. Benchmarking and monitoring framework for interconnected file synchronization and sharing services. **Future Generation Computer Systems**, 2017.

Nextcloud 12 Server Administration Manual. Disponível em: [https://docs.nextcloud.com/server/12/admin\\_manual/index.html](https://docs.nextcloud.com/server/12/admin_manual/index.html). Acesso outubro 2017.

NWOBODO, Ikechukwu (2015). A Comparison of Cloud Computing Platforms. **International Symposium on Circuits and Systems (ISCAS)**. Lisbon, Portugal, 24-27 May 2015. Atlantis Press.

OSANAIYE, Opeyemi et al. From cloud to fog computing: A review and a conceptual live VM migration framework. **IEEE Access**, 2017.

POPENTIU-VLADICESCU, Florin; ALBEANU, Grigore. Software reliability in the fog computing. In: **Innovations in Electrical Engineering and Computational Technologies (ICIEECT), 2017 International Conference on**. IEEE, 2017. p. 1-4.

PRINCY, S. Emima; NIGEL, K. Gerard Joe. Implementation of cloud server for real time data storage using Raspberry Pi. In: **Green Engineering and Technologies (IC-GET), 2015 Online International Conference on**. IEEE, 2015. p. 1-4.

Seafire Server Manual. Disponível em: <https://manual.seafire.com/overview/components.html>. Acesso outubro 2017.

STEINER, Wilfried; POLEDNA, Stefan. Fog computing as enabler for the Industrial Internet of Things. **e & i Elektrotechnik und Informationstechnik**, v. 133, n. 7, p. 310-314, 2016.

STOJMENOVIC, Ivan et al. An overview of Fog computing and its security issues. **Concurrency and Computation: Practice and Experience**, 2015.

VERBA, Nandor et al. Platform as a service gateway for the Fog of Things. **Advanced Engineering Informatics**, 2016.

WHAT IS DOCKER em: <https://www.docker.com/what-docker>. Acesso maio 2018.

ZHANG, Qi; CHENG, Lu; BOUTABA, Raouf. Cloud computing: state-of-the-art and research challenges. **Journal of internet services and applications**, v. 1, n. 1, p. 7-18, 2010.

ZHU, Jiang et al. Improving web sites performance using edge servers in fog computing architecture. In: **Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on**. IEEE, 2013. p. 320-323.

**ARTIGO PUBLICADO**

**ARTIGO 1**

Uma Revisão da Fog Computing e Seus Desafios de Pesquisas

**Journal on Advances in Theoretical and Applied Informatics – 2017**

# Uma Revisão da Fog Computing e Seus Desafios de Pesquisas

José dos Santos Machado, Edward David Moreno, Admilson de Ribamar Lima Ribeiro  
Departamento de Computação, Universidade Federal de Sergipe – UFS – Brasil  
jsmac18@hotmail.com, edwdavid@gmail.com, admilson@ufs.br

## Resumo

Este trabalho faz uma revisão do novo paradigma da computação distribuída que é a Fog Computing, apresenta seu conceito, características e áreas de atuação. Realiza uma revisão da literatura sobre a problemática da sua implementação e analisa seus desafios de pesquisas como: questões de segurança; questões operacionais e sua padronização. Percebemos que faltam muitas questões a serem investigadas no meio acadêmico para que sua implementação seja uma realidade, porém ficou evidente que sua adesão é inevitável para a internet do futuro.

Descritores: Fog Computing, Cloud Computing, Computação Distribuída, IoT, Integração Cloud à IoT, Desafios de Pesquisas.

## 1. Introdução

Nas últimas décadas, observa-se que serviços de computação como armazenamento, processamento de dados e controle foram transferidos para a “nuvem”. A oportunidade de computação ilimitada na nuvem permite que os usuários finais acessem amplas informações facilmente, também é possível visualizar que os dispositivos móveis e sensores, como smartphones, se tornaram poderosos equipamentos, o que levou ao surgimento de novos sistemas e aplicações.

Estes sistemas e aplicações introduzem novas demandas funcionais em computação e redes que a “nuvem” sozinha não pode atender. Neste cenário percebe-se que a computação local na borda da rede é muitas vezes necessária [6]. Por exemplo, para processar dados em tempo real, criar contextos de reconhecimento de localização a partir de sensores locais e maximizar a eficiência das comunicações sem fio na borda da rede. No entanto, a nuvem está muito longe dos dispositivos para satisfazer requisitos de latência e, é muito centralizada para lidar com a heterogeneidade e diversidade contextual em uma área local, também é muito custoso carregar todos os dados de sensores individuais para a nuvem [28].

Para ultrapassar estas limitações, porções da capacidade de computação da nuvem podem ser deslocados para a borda da rede e formam um ambiente de computação local, isto é, uma “Fog Computing” chamada também de “nevoeiro”. Ao distribuir os serviços de computação e de rede mais

próximos de onde os dados do usuário são gerados, a Fog atende melhor às demandas emergentes [21].

A *Fog Computing* apresenta uma nova arquitetura que “leva processamento para os dados”, enquanto a nuvem “leva os dados para o processamento” [1]. Dessa maneira dispositivos de borda e dispositivos móveis podem estar interligados dentro de uma rede local e executar colaborativamente tarefas de armazenamento, processamento de dados de rede e de controle [11].

Em uma arquitetura de *Fog Computing*, redes de sensores podem desempenhar um papel significativo na medida em que sensores e atuadores implantados em ambiente distribuído podem vir a ser geradores de dados, controladores para sistemas físicos e plataformas de computação em rede. A *Fog* terá *gateways* de borda que têm mais capacidade de computação. Uma rede de sensores, incluindo sensores e atuadores, estará profundamente conectada aos *gateways* do “nevoeiro” e aos dispositivos móveis diretamente e, em seguida, faz extensas interações com eles, isso fornecerá novos serviços que conectam o ambiente físico à infraestrutura cibernética.

A *Fog Computing* pode vir a resolver muitos problemas da Internet das Coisas (IoT), por exemplo, os serviços da *Fog* serão capazes de melhorar a largura de banda e as restrições de custo das comunicações de longo alcance. No entanto, muitos desafios ainda permanecem na computação em *Fog*, como modelar uma arquitetura de sistema para a *Fog*; como implementar, organizar e gerenciar dispositivos da *Fog*; como a *Fog* interage com a nuvem e com os dispositivos; e como gerenciar a conectividade física e lógica da *Fog*; entre outros.

Este trabalho apresenta uma revisão do novo paradigma da computação distribuída que é a *Fog Computing*, seu conceito, características e áreas de atuação são apresentados. Faz uma revisão da literatura sobre a problemática da sua implementação e analisa seus desafios de pesquisas.

O artigo está organizado em cinco seções, a seção 2 apresenta o conceito e características da *Fog Computing*, a seção 3 é dedicada a revisão da literatura, na seção 4 temos os desafios de pesquisas e por fim na seção 5 as conclusões e trabalho futuro.



## 2. Fog Computing

Nesta seção são apresentados e explicados o conceito, as características e as áreas de aplicação e atuação do novo paradigma da computação distribuídas que é a *Fog Computing*.

### 2.1 Conceito da Fog Computing

Grande parte dos trabalhos tem focado em como efetivamente descarregar tarefas computacionais intensivas de dispositivos com recursos limitados para a nuvem e obter os resultados desejados [6].

No entanto, devido à latência de rede frequentemente imprevisível, especialmente em um ambiente móvel, muitas vezes a computação em nuvem não pode atender aos requisitos rigorosos de latência, segurança e privacidade dos aplicativos em área restrita geograficamente [31]. Por outro lado, a crescente quantidade de dados gerados por dispositivos e sistemas, com poucos recursos pode se tornar impraticável para transportar dados através de redes para nuvens remotas [24].

Para isso, surgiu um novo diagrama, a *Fog Computing*. O conceito de computação em *Fog* foi adotado pela *Cisco Systems* como um novo paradigma em 2012 [11]. A *Fog Computing* é a computação em nuvem que distribuirá serviços avançados de computação, armazenamento, rede e gerenciamento mais próximos dos usuários finais, enviando informações dos dispositivos IoT para *Cloud Computing*, formando assim uma plataforma distribuída e virtualizada, assim, também é referido como computação de borda [14]. A figura 1 mostra a localização entre *Fog Computing* e *Cloud Computing*.

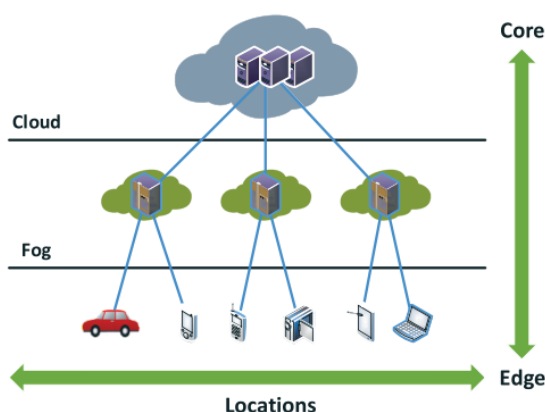


Fig. 1 - Fog localizada entre borda e nuvem [28]

A computação em *Fog* oferece muitas vantagens desejadas pelas aplicações de hoje, como processamento em tempo real, rápida escalabilidade e compartilhamento de recursos e armazenamento local. Como tal, a computação em *Fog* rapidamente conquistou muita atenção da indústria e da academia. É naturalmente a ponte entre a Internet das Coisas

(IoT) e a computação em nuvem, com a infraestrutura de computação existente na Internet [17].

### 2.2 Características da Fog Computing

A computação *Fog* é um paradigma inovador que realiza computação distribuída, serviços de rede e armazenamento, além da comunicação entre *Cloud Computing Data Centers* até os dispositivos ao longo da borda da rede. Essa comunicação amplia as operações e serviços inerentes à computação em nuvem, permitindo assim uma nova geração de aplicativos [24]. A principal função é filtrar e agregar dados para os centros de dados da *Cloud* e aplicar inteligência lógica a dispositivos finais. A figura 2 apresenta arquitetura da *Fog Computing* com a sua localização.

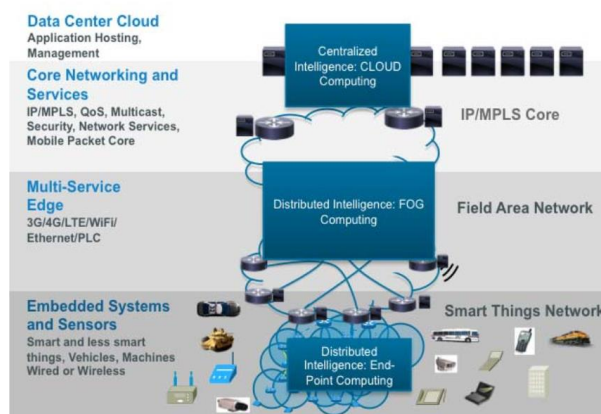


Fig. 2 - Arquitetura da Fog Computing [31]

Devido à sua recente introdução e emergência, não há nenhuma arquitetura padrão disponível em relação ao gerenciamento de recursos baseado em *Fog*, e por esse motivo ainda existe um modelo simples para esse propósito, considerando a previsão de recursos, a alocação de recursos e os custos de forma realista e dinâmica, considerando também o tipo, e características dos clientes, este modelo é adaptável às exigências de diferentes clientes.

As principais características da *Fog Computing* são:

- **Heterogeneidade:** A *Fog Computing* é uma plataforma virtualizada que oferece serviços computacionais, de rede e de armazenamento entre a computação em nuvem e dispositivos finais de diferentes tipos e formas;
- **Distribuição geográfica:** A computação em *Fog* possui uma implementação amplamente distribuída para oferecer serviços de alta qualidade para dispositivos finais móveis e fixos;
- **Localização de borda, percepção de localização e baixa latência:** O conceito de computação em *Fog*

foi implementado para suprir a falta de suporte para pontos finais, com serviços de qualidade à beira da rede.

- **Interação em tempo real:** Várias aplicações de *Fog*, como sistemas de monitoramento de tráfego, exigem processamento em tempo real em vez de processamento em lote;
- **Suporte mobilidade:** O suporte à mobilidade é essencial para muitos aplicativos de computação em *Fog*, para permitir a comunicação direta com dispositivos móveis usando protocolos como o Protocolo de separação de localização / ID da Cisco, que desacopla a identidade do host da identidade de localização usando um sistema de diretório distribuído;
- **Grande escala de redes de sensores:** Isso é aplicável ao monitorar o ambiente ou em rede inteligente, usando sistemas inerentemente distribuídos que requerem computação distribuída ou recursos de armazenamento;
- **Prevalente para acesso sem fio:** A maioria dos pontos de acesso sem fio e o *gateway* de dispositivos móveis são exemplos típicos de um nó de *Fog* na rede;
- **Interoperabilidade:** Os componentes de *Fog* devem ser capazes de interoperar para garantir suporte para ampla gama de serviços, como transmissão de dados.

*Fog Computing* é semelhante à computação em nuvem em muitos aspectos, no entanto, pode ser diferenciado do anterior pelo fato de estar próximo dos dispositivos finais, a distribuição espacial geograficamente menor e a possibilidade de apoio à mobilidade [7]. Como o processamento baseado em *Fog* ocorre ao longo da borda da rede, os resultados finais refletem uma percepção de localização altamente melhorada, baixa latência e Qualidade de Serviço (QoS), em aplicações de *streaming* e tempo real, os nós de nevoeiro são dispositivos heterogêneos, que vão desde servidores, pontos de acesso, roteadores de borda até dispositivos finais como telefones celulares, relógios inteligentes, sensores e etc [1].

A *Fog* funciona como um link entre IoT e a Nuvem para introduzir as funcionalidades extras necessárias para o processamento específico da aplicação, como filtragem e agregação. Antes de transferir os dados para a Nuvem, deve ser capaz de decidir o que deve ser enviado (o conteúdo), como (formato de dados) e quando enviar (tempo). Durante esse processo, ela também precisa excluir alguns dados redundantes ou inválidos, e agregar os dados complementares no espaço e no tempo dimensionado [1]. A tabela 1 mostra a comparação de resultados entre a *Cloud* e a *Fog*.

Aspectos Investigados	Cloud Tradicional	Fog Computing
Previsão de latência	5 segundos	1,5 segundos
Latência de exibição de página da Web	8 segundo	3 segundos
Tráfego de internet	75 Kbps	10 Kbps
Hardware usado	Amazon Web Server	Sistemas embarcados, a exemplo do Raspberry PI

Tabela 1 - Resultados entre a Cloud e a Fog [1]

## 2.3 Áreas de aplicação e atuação

Diferentes aplicações de computação em *Fog* foram sugeridas na literatura. Segundo OSANAIYE *et al.* (2017) [24], as categorias das aplicações de computação em *Fog* são divididas em: (i) Aplicações em tempo real; (ii) Aplicações quase em tempo real; (iii) Aplicações introduzidas em redes.

- As aplicações em tempo real são aplicações de baixa latência, que funcionam dentro de um período de tempo pré-definido, sendo classificada pelo usuário como imediata ou urgente.
- Quase em tempo real, por outro lado, são aplicações que estão sujeitas a atraso de tempo introduzido pelo processamento de dados ou transmissão de rede, entre o momento em que ocorre um evento e o uso dos dados para processamento.
- A computação em *Fog* também pode ser introduzida em uma rede (para aplicações que não necessitem de processamento e transmissão em tempo real) para reduzir a quantidade de tráfego no núcleo de processamento.

Vamos apresentar algumas áreas de aplicação e atuação da *Fog Computing*, disponibilizada na literatura e que poderá ser beneficiada com a sua implementação.

### a) Vídeo streaming

A transmissão de aplicativos e serviços de vídeo aproveitará alguns dos principais benefícios da computação em *Fog*, que incluem conhecimento de localização, baixa latência, suporte para mobilidade e análise em tempo real. Vários dispositivos inteligentes suportam uma aplicação de vigilância inteligente, que pode ser usada por agentes da lei para exibir fluxos de eventos em vídeos de suspeitos em tempo real.

### b) Jogos

O advento da computação em nuvem forneceu uma plataforma para jogos de computador sem

usuários (jogadores) preocupados com os requisitos de *hardware*. Os provedores de jogos em nuvem nos últimos tempos têm expandido rapidamente a infraestrutura da nuvem para fornecer o serviço de jogos sob demanda (GoD) aos usuários através da Internet. É oferecido remotamente, permitindo um jogo interativo que pode ser acessado e decodificado por dispositivos finais, como *smartphones* ou *tablets*.

#### **c) Cuidados de saúde**

As aplicações IoT forneceram uma abordagem estruturada para melhorar os serviços de saúde. Isso é conseguido através da implantação de sistemas de monitoramento onipresentes e transmissão dos dados para dispositivos de *Fog* em tempo real, antes de enviar a informação para a nuvem para posterior análise e diagnóstico.

#### **d) Sistema inteligente de semáforo (STLS)**

O semáforo inteligente foi idealizado por veículos inteligentes conectados e sistema de transporte avançado. Ele interage localmente com uma série de nós sensores para detectar a presença de ciclistas, motociclistas ou pedestres, também estima a velocidade e a distância dos veículos que se aproximam. Esta informação pode ser usada para prevenir acidentes ao enviar sinais de aviso prévio aos veículos que se aproximam.

#### **e) Cidades inteligentes**

Uma cidade inteligente é a chave essencial para aplicação IoT, que varia desde o gerenciamento inteligente de tráfego, até o gerenciamento de energia de edifícios. O conceito de cidade inteligente atraiu o grande interesse dos setores de ciência, engenharia, de comunidades de pesquisa e de profissionais, como um meio para superar os desafios associados ao rápido crescimento urbano.

#### **f) Veículos inteligentes**

O advento da computação em nuvem para dispositivo móvel exigiu o estudo de seus agentes, como veículos, robôs e seres humanos, que interagem juntos para detectar o meio ambiente, processar os dados e transmitir os resultados. Veículo conectado através da *Fog Computing* poderá se comunicar com seu ambiente interno e externo.

#### **g) Smart Grid**

Os aplicativos de balanceamento de carga de energia podem ser executados em dispositivos de borda de rede, como medidores inteligentes e micro grades. Com base na demanda de energia e na possibilidade de um custo mais baixo, esses dispositivos alternam automaticamente para energias alternativas, como a energia solar e o vento.

#### **h) Redes de sensores e atuadores sem fio**

As redes de sensores sem fio tradicionais são insuficientes em aplicações que vão além de detecção e rastreamento, que exigem que os atuadores exerçam ações físicas como abrir, fechar ou mesmo movimento de sensores. Nesse cenário, os atuadores que atuam como dispositivos de *Fog* podem controlar o próprio processo de medição, a estabilidade e os comportamentos oscilatórios criando um sistema de circuito fechado.

#### **i) Controle de construção inteligente**

As aplicações deste cenário são facilitadas por sensores sem fio implantados para medir temperatura, umidade ou níveis de vários gases na atmosfera de construção. Neste caso, as informações podem ser trocadas entre todos os sensores em um piso, e suas leituras podem ser combinadas para formar medições confiáveis. Os sensores irão usar a tomada de decisão e a ativação de dispositivos de *Fog* para reagir com as informações dos dados recebidos.

#### **j) IoT e sistemas ciberfísicos (CPSs)**

Os sistemas baseados em *Fog Computing* estão se tornando uma classe importante de IoT e CPSs. Com base nas operadoras de informação tradicionais, incluindo Internet e rede de telecomunicações, a IoT é uma rede que pode interconectar objetos físicos comuns com endereços identificados. Os CPSs apresentam uma combinação dos elementos computacionais e físicos do sistema.

#### **k) Redes Definidas por Software (SDN)**

A estrutura de computação em *Fog* pode ser aplicada para implementar o conceito SDN para redes de veículos. SDN é um paradigma emergente de computação em rede e tornou-se um dos tópicos mais populares na indústria de TI. Ela separa as camadas de controle e comunicação de dados. O controle é realizado em um servidor centralizado e os nós seguem o caminho de comunicação decidido pelo servidor.

#### **l) Trem autossustentável**

Automanutenção de trens é uma importante área de aplicação. As variações de temperatura podem ser detectadas pelo sensor de monitoramento de rolamento de esferas no trem e podem notificar automaticamente qualquer tipo de transtorno. Isso ajudará na prevenção de desastres.

As aplicações atuais e futuras que exigem a computação em *Fog* incluem veículos conectados, veículos de piloto automático, redes inteligentes, redes de sensores e atuadores sem fio, casas inteligentes, cidades inteligentes, sistemas conectados de petróleo e gás, e sistemas móveis de saúde [6].

### 3. Revisão da literatura

A *Cloud Computing* é uma forma de computação baseada na Internet que pode fornecer recursos de *hardware* e *software*, de acordo com as necessidades. *Cloud Computing* virtualiza a infraestrutura de computação, redes, armazenamento e outros, para formar uma grande gama de recursos compartilhados, mensurável e dinâmico, e fornece um modelo de computação para todos os tipos de usuários na forma de serviços controlados através de uma plataforma de gestão [3,4,5,9,13,30].

A Internet das Coisas (IoT) são os dispositivos inteligentes conectados que tiveram um crescimento exponencial em termos de tecnologias, participação no mercado e aprovação dos consumidores, pavimentando o caminho para a evolução dos princípios de computação de *Fog*, melhorando gradualmente oportunidades produtivas em vários domínios como redes de veículos e o *Smart Grid*, as vantagens deste procedimento de computação para serviços em vários domínios são necessários e devem ser investigados.

*Fog Computing* permite maior suporte e melhor tempo de resposta à Internet das Coisas, é adequada para solicitações de serviço em tempo real, e para compartilhar recursos de forma eficiente, uma eficiente e cooperativa utilidade baseada em estratégia de emparelhamento é necessária entre os nós do IoT [27].

#### 3.1 Revisão Sistemática

Para alcançar um grau de rigor científico, procurou-se assegurar o processo de investigação a partir das concepções sobre Revisão Sistemática (RS) conforme descreve-se a seguir.

A revisão sistemática de literatura, identifica, avalia e interpreta todas as pesquisas disponíveis relevantes para uma questão específica, área temática, ou fenômeno de interesse [15].

A importância do estabelecimento de um processo de revisão sistemática se concretiza ao observar a definição de fases, quais sejam: a) Planejamento da revisão; b) Procedimentos de condução e extração da RS e, ao final, c) Procedimentos de elaboração de relatórios da RS.

Dessa forma, os resultados são mais confiáveis em relação à revisão de literatura primária, em virtude de sua forma rigorosa e que dá possibilidade de repetições e auditoragem.

Assim, a RS teve o seu foco na caracterização dos processos de desenvolvimento de solução para implementação da *Fog Computing*.

Foram identificados 825 artigos (gráfico 1), dos quais, em cada base totalizaram: ACM (51 artigos), GOOGLE SCHOLAR (21 artigos), IEEE (294 artigos), SCIENCE DIRECT (189 artigos), SCOPUS (130 artigos) e SPRINGER (140 artigos).

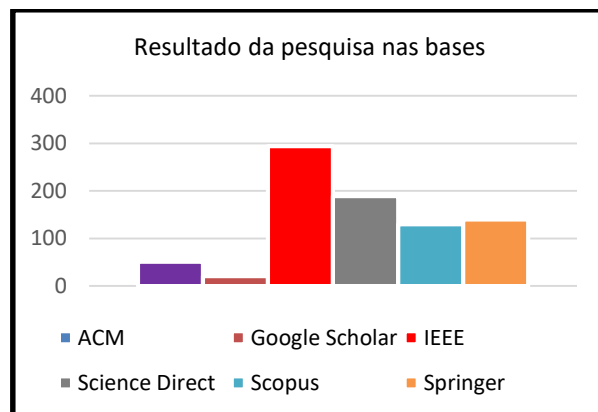


Gráfico 1 - Artigos identificados

Fonte: Própria do autor (2017)

Nesse momento, houve o julgamento dos critérios de exclusão dos 825 trabalhos relacionados no processo de identificação, 212 trabalhos foram aceitos para ser avaliados na etapa de extração da revisão de literatura, assim 613 foram excluídos, sendo 355 pelos critérios definidos no protocolo de exclusão (trabalhos que não apresentaram texto completo; trabalhos onde os termos de busca não se apresentam nos campos: título, resumo e palavras-chave e trabalhos de conclusão de curso) e 258 trabalhos excluídos (gráfico 2) por estar duplicados quando da exportação dos resultados de consultas das bases.

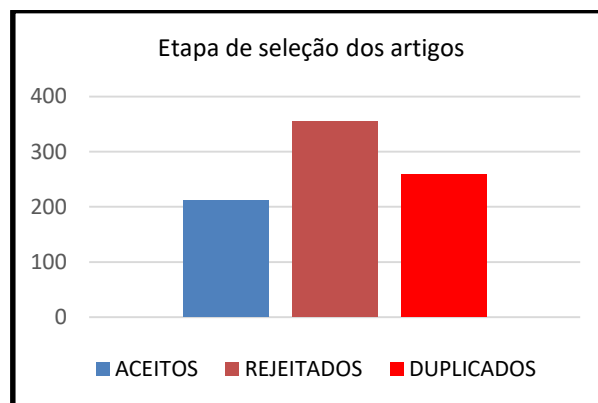


Gráfico 2 - Resultado da etapa de seleção dos artigos

Fonte: Própria do autor (2017)

A fim de estabelecer os interesses da pesquisa, o protocolo construído definiu os seguintes critérios de seleção de trabalhos: artigos publicados no período de 2010 a 2017; artigos completos disponíveis *online*; idioma inglês e português; estudos que abordem áreas da *Fog Computing* e estudos que abordem a integração da *Cloud* à IoT.

Os 212 trabalhos aceitos por, pelo menos, um critério de seleção e conduzidos à etapa de extração para o processo de leitura completa, onde conclui-se a condução de 30 trabalhos, os quais atenderam os critérios formalizados no protocolo da revisão

sistemática e fazem parte das minhas referências bibliográficas.

### 3.2 Trabalhos analisados

Um total de oito artigos foram identificados na implementação da *Fog Computing*, para melhorar alguns dos serviços da integração entre a *Cloud* e os dispositivos IoT, apresentaremos seus resultados e seus respectivos trabalhos futuros de pesquisas.

ZHU *et al.* (2013) [31], apresentaram a otimização da web dentro do contexto *Fog Computing*. Aplicaram métodos existentes para otimização da web de uma maneira inovadora, de tal forma que esses métodos podem ser combinados com conhecimento exclusivo que está disponível apenas nos nós de borda (*Fog*). Como trabalho futuro sinalizaram aplicar seus conceitos propostos para desenvolver um sistema de prova de conceito.

No trabalho de CRACIUNESCU *et al.* (2015) [8], apresentaram uma implementação em laboratório de e-Saúde, onde o processamento em tempo real é realizado pelo PC doméstico, enquanto os metadados extraídos são enviados para a nuvem para processamento posterior. Como trabalho futuro sinalizaram adicionar mais sensores e mais dispositivos *off-the-shelf*, que são atualmente *mainstream*, e os dados de fusão desses dispositivos.

Em [29] VERBA *et al.* (2016), analisaram as plataformas existentes e suas deficiências, bem como propor uma plataforma de *gateway* modular, baseada em mensagens que permite o agrupamento de *gateways* e a abstração dos detalhes do protocolo de comunicação periférica. E sinalizaram como trabalho futuro desenvolver um ambiente de laboratório inteligente com diversos dispositivos mais complexos e cenários de controle para testar completamente o sistema.

FAN *et al.* (2016) [10], apresentaram a capacidade do recurso de *Web Caching* adicionado ao dispositivo de borda para servir como servidor proxy de armazenamento em cache, para obter mais armazenamento em cache. Os dispositivos finais também são explorados para fornecer algum espaço de cache. Como trabalho futuro sinalizaram adicionar mais funcionalidades ao dispositivo de borda, como a segurança e implementar o trabalho no mundo real.

No trabalho de HAO *et al.* (2017) [12], apresentaram um design do WM-FOG, uma estrutura de computação para ambientes *Fog*, que engloba essa arquitetura de *software* e avalia seu protótipo do sistema. Como trabalho futuro sinalizaram adicionar mais recursos ao WM-FOG para melhor atender às aplicações de computação em *Fog*.

Em [17] LI *et al.* (2017), discutiram dois conceitos de codificação recentemente propostos, códigos de mínima largura de banda e códigos

mínima latência, e ilustram seus impactos na computação em *Fog*, também analisaram uma estrutura de codificação unificada que inclui as duas técnicas de codificação acima descritas. Como trabalho futuro sinalizaram a necessidade de pesquisar computação heterogênea; redes com topologia de camada múltipla e estruturada; tarefas de computação em várias etapas; custos de computação codificados; verificar a computação distribuída; explorar as estruturas algébricas das tarefas computacionais; aplicações pesadas de comunicação e nós de *Fog plug-and-play*.

OSANAIYE *et al.* (2017) [24], apresentaram uma abordagem conceitual de migração em tempo real de pré cópia para a migração de VM e sinalizou como trabalho futuro a implantação do *framework* no mundo real ou ambiente de teste, com o objetivo de sua validação.

E por fim, POPENTIU-VLADICESCU *et al.* (2017) [25], analisaram os modelos de arquiteturas e práticas existentes em *Fog Computing* visando a confiabilidade e segurança dos sistemas de *Fog*, uma abordagem considerada integradora de três componentes da confiabilidade do sistema: a confiabilidade dos nós, a confiabilidade da rede e a confiabilidade do *software*, a arquitetura de referência *OpenFog* e os esquemas AJIA e BDSC. Como trabalho futuro sinalizaram resolver problemas técnicos e de algoritmos no paradigma de *Fog Computing*.

A análise dos estudos apresentados permitiu identificar que, a produção científica sobre a implementação da *Fog Computing* ainda é escassa. Desse modo, nota-se a necessidade de maiores investimentos em pesquisa nesta temática, dada a importância da implementação da *Fog Computing* como tecnologia que promoverá segurança e velocidade na transmissão dos dados entre a *Cloud* e IoT.

Artigo	Fog Computing	Plataforma Cloud	Dispositivo Embarcado	Benchmark
[31] ZHU (2013)	✓			
[8] CRACIUNESCU (2015)	✓	✓	✓	
[29] VERBA (2016)	✓	✓	✓	✓
[10] FAN (2016)	✓			✓
[12] HAO (2017)	✓	✓		✓
[17] LI (2017)	✓			✓
[24] OSANAIYE (2017)	✓			
[25] POPENTIU-VLADICESCU (2017)	✓			

Tabela 2 – Comparação entre os trabalhos

Fonte: Própria do autor (2017)

Na tabela 2 os dados dos artigos analisados são sumarizados e apresentados, foram organizados na

ordem crescente cronologicamente para demonstrar a evolução em relação ao tema *Fog Computing*. Os artigos foram comparados quanto a implementação da *Fog Computing*, a utilização de plataforma de *Cloud*, utilização de dispositivo embarcado e uso de técnicas ou métodos de avaliação (*Benchmark*).

## 4. Desafios de pesquisas

*Fog Computing* é uma extensão da clássica *Cloud Computing* para a borda da rede (como a névoa é uma nuvem perto do chão). Ela foi projetada para suportar aplicações da Internet das coisas (IoT) caracterizadas por limitações de latência, exigência de mobilidade e de distribuição geográfica. Apesar da computação, armazenamento e rede serem recursos tanto da *Cloud* e da *Fog*, o último tem características específicas, localização na borda e reconhecimento do local implicando baixa latência, distribuição geográfica e um grande número de nós em contraste com a nuvem centralizada, apoio à mobilidade (através de acesso sem fio), interação em tempo real (em vez de processos batch) e suporte para interação com a *Cloud* [10,14].

Em [5] os autores fizeram uma análise que mostra como a construção de projetos de *Fog Computing* é um desafio, de fato, a adoção de abordagens baseadas em *Fog* requer vários algoritmos e metodologias relacionadas com a confiabilidade das redes de dispositivos inteligentes, e que operam sob condições específicas que pedem técnicas de tolerância a falhas específicas.

A *Fog Computing* é proposta para resolver os problemas acima mencionados. Como a *Fog Computing* é implementada na borda da rede, ela fornece baixa latência, definição geográfica de localização e melhora a qualidade de serviço (QoS) para aplicações que usam *streaming* e aplicações em tempo real [16]. Exemplos típicos incluem automação industrial, transporte, redes de sensores e atuadores. Além disso, essa nova infraestrutura suporta heterogeneidade, pois os dispositivos *Fog* incluem dispositivos para usuários finais, pontos de acesso, roteadores e *switches*. Conforme afirma-se em [28], o paradigma *Fog* está bem posicionado para análise de dados em tempo real, suporta pontos de coleta de dados densamente distribuídos e oferece vantagens em entretenimento, publicidade, computação pessoal e outras aplicações.

### 4.1 Questões de Segurança

Como uma "pequena nuvem" perto dos usuários finais, os dispositivos de *Fog* podem enfrentar desafios de segurança de sistema e deverão ser implementados em locais de vigilância e proteção rigorosas. A fim de proteger dos objetivos de mal-intencionados, como a espionagem e o sequestro de dados. Os ataques tradicionais tornam-se disponíveis para comprometer o sistema de dispositivos de *Fog*

[28]. Nesta seção, discutiremos os problemas de segurança do sistema em *Fog Computing*.

#### a) Técnicas de detecção de intrusão

As técnicas de detecção de intrusão também podem ser aplicadas na computação de *Fog*. A intrusão em redes inteligentes pode ser detectada usando um método baseado em assinatura, no qual os padrões de comportamento são observados e verificados em um banco de dados. A intrusão também pode ser capturada usando um método baseado em anomalia em que um comportamento observado é comparado com o comportamento esperado para verificar se há um desvio de conduta.

Em [28], os autores apresentam um exemplo de ataque de homem no meio. O ataque homem no meio tem potencial para se tornar um ataque típico na computação em *Fog*. Neste ataque, os *gateways* que servem como dispositivos de *Fog* podem ser comprometidos ou substituídos por falsos. Exemplos são os clientes KFC ou da *Star Bar* que se conectam a pontos de acesso maliciosos que fornecem identificadores de conjunto de serviços enganosos como legítimos. A comunicação privada das vítimas será sequestrada uma vez que os atacantes assumam o controle dos *gateways*.

#### b) Autenticação e autorização

As questões de autenticação e autorização não foram estudadas no contexto da computação em *Fog*. Elas foram estudadas no contexto de redes inteligentes e comunicação máquina a máquina [28]. Existem soluções de segurança para computação em nuvem. No entanto, elas podem não se adequar à computação em *Fog*, porque os dispositivos de *Fog* funcionam na extremidade das redes. O ambiente de trabalho dos dispositivos *Fog* enfrentará muitas ameaças que não existem em uma nuvem bem gerenciada.

Os dispositivos de *Fog* geralmente possuem algum tipo de conectividade com o servidor de autenticação remoto da Nuvem, que pode ser usado para distribuir informações de autenticação e coletar logs de auditoria, mas essa conectividade pode ser muito lenta em certos ambientes.

Além disso, a dependência dos servidores de autenticação central da *Cloud* não é o ideal, porque a autenticação deve continuar a ser aplicada para a pessoa que acessa os dispositivos localmente, quando a comunicação do servidor de autenticação remota estiver desativada. Uma provisão para garantir que o acesso necessário esteja disponível em situações de emergência pode ser importante, mesmo que isso signifique ignorar o controle de acesso normal, mas com um certo controle de auditoria.

#### c) Segurança de rede

O desenvolvimento de tecnologia sem fio levou ao aumento de problemas de segurança. A

computação em *Fog* é afetada de maneiras semelhantes, como outras tecnologias sem fio. Vários exemplos de tais ataques são *sniffer*, *spoofing*, *jamming*, etc. Esses ataques normalmente ocorrem entre o nó de *Fog* e o sistema de nuvem centralizado. Geralmente, costuma-se confiar nas configurações definidas pelo sistema central da rede, que separa o tráfego de dados normal da rede do cliente. Por este motivo sobrecarrega muito o gerenciador de rede. Além disso, os nós de neblina estão na borda da rede, o que aumenta o processamento do gerenciador de rede. SDN (*Software Defined Networking*) pode ser usado como uma abordagem para o gerenciador de rede para trabalhar no baixo nível de abstração para os serviços de rede [16]. Pode ajudar no gerenciamento, aumentar a escalabilidade da rede e reduzir o custo com referência à computação em *Fog*.

#### d) Segurança dos dados

Na *Fog Computing* o controle do usuário para os dados é transmitido pelo nó de *Fog*, daí os mesmos problemas de segurança surgem da computação em nuvem. A integridade dos dados não poderá ser mantida quando os dados forem perdidos ou também modificados. Os dados que são carregados para o nó de *Fog* também podem ser usados por terceiro. Existem várias técnicas que podem ser usadas para fornecer integridade de dados, confidencialidade e veracidade, como a combinação de criptografia homomórfica e criptografia *searchable* [16]. Essas técnicas garantem que o cliente não armazene dados no servidor não confiável. Também os dados podem ser recuperados de maneira muito mais rápida, e portanto, o custo da comunicação se torna baixo. Sempre haverá novos desafios na computação em *Fog*, relacionados ao sistema de armazenamento que pode lidar com as operações dinâmicas de uma maneira muito mais rápida e ocupar menos espaço.

#### e) Privacidade

A privacidade do usuário é fator primordial em qualquer sistema de computação, essa questão é de grande preocupação também na computação em *Fog*. Não só o prestador de serviços, mas o fato de que o governo também está envolvido torna-o mais desafiador. Na verdade, torna-se ainda mais fácil para o terceiro ter acesso aos dados privados do usuário, localização e etc. E portanto, o usuário se torna propenso a ataques. Devido à entrega de dados na borda do nó de *Fog*, torna-se muito mais fácil coletar todas as informações importantes dos usuários. Este é um dos problemas mais desafiadores na *Fog Computing*, preservar a privacidade dos usuários [16].

#### f) Protocolos seguros e eficientes

Muitos protocolos existentes, são baseados em tempo sincronização e transmissões de pacotes em redes sem fio e não são adequados para dispositivos

IoT com recursos limitados em computação em *Fog*. As transmissões sem fio e os cálculos de segurança consomem uma fração significativa da capacidade de energia [2]. O principal desafio é como projetar esquemas seguros e eficazes no IoT sem sacrificar o desempenho e consumir uma alta taxa energia.

#### g) Verificação de localização

Em ambientes hostis como veículos e ferrovias, os dispositivos IoT em um ambiente de *Fog* podem se mover de forma rápida e dinâmica, o que dificulta a verificação da localização. O principal desafio é como projetar um esquema de verificação de localização seguro e preciso em ambientes tão volátil e, ao mesmo tempo, o esquema deve ser adequado para dispositivos IoT com recursos limitados.

## 4.2 Questões Operacionais

Em [12], os autores apresentam uma série de questionamentos sobre o funcionamento da *Fog Computing*, como questões a serem pesquisadas e definidas.

- (i) Para nós de *Fog* com *hardware* heterogêneo, é aceitável trocar eficiência energética por latência reduzida?
- (ii) Um processo em um nó de *Fog* deve ser interrompido quando o usuário se mover em direção a outro nó de *Fog*?
- (iii) Como as tarefas devem ser agendadas considerando latência, consumo de energia, mobilidade e carga de trabalho existente?
- (iv) Onde o programa de agendamento deve ser executado?
- (v) Quais são os benefícios das tarefas conjuntas de agendamento?
- (vi) Quando os serviços devem ser iniciados e interrompidos?
- (vii) Qual a melhor forma de equilibrar a carga de trabalho?
- (viii) A virtualização que deverá ser usada será baseada em VM ou baseada em contêiner?
- (ix) Quais metodologias devem ser usadas para provisionar serviços eficientemente para centenas, milhares e milhões de usuários?
- (x) Qual é a melhor maneira de dividir a carga de trabalho, para serviços que agregam informações de dispositivos clientes próximos em relação à eficiência energética do lado do cliente?

Na computação em nuvem, a consistência dos dados pode ser conseguida coordenando os servidores da nuvem nos centros de dados onde a nuvem é implantada. Na computação em *Fog*, no entanto, as coisas se tornam complicadas. Ao implementar objetos de dados em um ambiente de *Fog*, é necessário não só coordenar os servidores de



nuvem de back-end, mas também validar os dados em cache nos nós de Fog, bem como nos dispositivos clientes, se for necessária uma forte consistência de dados [12]. Isso pode resultar em um desempenho de gravação deteriorado, o que enfraquece os benefícios do uso de nós de Fog como os servidores de cache de gravação. Por outro lado, a computação em Fog também oferece oportunidades para alcançar a consistência dos dados de forma mais eficiente do que a computação em nuvem.

### 4.3 Padronização

A Cisco Systems tenta padronizar a computação em Fog e criou a OpenFog Consortium para promover sua visão. O que quer que seja chamado e definido, a chave é reduzir a latência para aplicativos críticos, movendo a transferência de dados e o processamento para a borda da rede, mais perto do dispositivo IoT [18].

Podendo ser aplicado em dezenas de sistemas onde os dados e aplicativos foram implementados próximo da borda da rede, mas ainda funcionam com dados e aplicativos centralizados. Embora seja um pouco complicado, não é tão difícil de fazer. A OpenFog beneficiará em muitos fatores a adesão da Fog Computing, como:

- Uma arquitetura padrão e tecnologia de habilitação que permite abordar a computação de *Fog* de forma simples, mas consistente;
- A capacidade de fornecer uma boa estrutura de desenvolvimento de produto que os dispositivos de rede e construtores de *software* possam seguir, bem como influenciar seu desenvolvimento;
- A capacidade de lidar com a segurança de forma consistente. Os ataques DDOS assumiram dispositivos, computadores, e agora que tudo é inteligente e possui um sistema operacional, esse será um fator crítico da internet do futuro.

O *OpenFog* publicou recentemente uma arquitetura de referência para padronizar a implementação da Fog Computing, alicerçada em oito pilares para abranger uma arquitetura padrão considerando aspectos de segurança, escalabilidade, aberta, autônoma, parceira, ágil, hierárquica e de programabilidade (ver a figura 3).

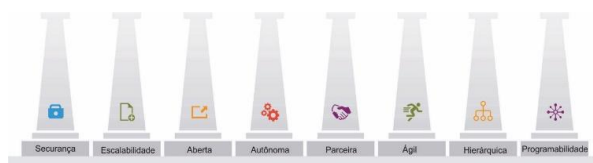


Fig. 3 - Arquitetura de referência OpenFog [18]

É necessária uma arquitetura aberta para desenvolver um ambiente de vida inteligente com potencialmente milhares de fornecedores [21]. O

Consórcio OpenFog é formado pelas empresas EDGE Lab da Cisco, Dell, Intel, Microsoft e Princeton para acelerar a adoção da computação em Fog ([www.openfogconsortium.org](http://www.openfogconsortium.org)) e está tomando medidas para criar uma arquitetura de referência comum, que cobre plataforma de *hardware* e *software* com recursos altamente sofisticados.

## 5. Conclusão e trabalho futuro

A *Fog Computing* poderá distribuir serviços avançados de computação, armazenamento, rede e gerenciamento mais próximos dos usuários finais ao longo da borda da rede, entre à *Cloud*, IoT e dispositivos móveis, formando assim uma plataforma distribuída e virtualizada.

Problemas de latência, armazenamento e poder de processamento desses dispositivos fizeram surgir a *Fog Computing*. Assim, a computação em *Fog* rapidamente atraiu muita atenção da indústria e da academia.

Este trabalho realizou uma revisão do novo paradigma da computação distribuída que é a *Fog Computing*, apresentou seu conceito, características e áreas de atuação. Fez uma revisão da literatura sobre a problemática da sua implementação e analisou seus desafios de pesquisas como: questões de segurança; questões operacionais e sua padronização.

Percebemos que faltam muitas questões a serem investigadas no meio acadêmico, para que sua implementação seja uma realidade, porém ficou evidente que sua adesão é inevitável para a internet do futuro.

Para novos trabalhos futuros pretendemos propor a implementação e análise de uma Fog Computing, para fornecer StaaS (Storage as a Service), a dispositivos IoT utilizando uma plataforma embarcada baseada em Raspberry PI. Podendo ser uma boa alternativa para sanar problemas relacionados ao armazenamento de dados dos dispositivos IoT, servindo como Fog Computing para armazenar dados desses dispositivos e enviar à Cloud Computing, com segurança e eficiência no transporte dos dados entre as redes. Utilizando sistemas embarcados em plataforma de dispositivo de baixo custo, em vez de usar potentes e caros servidores para exercer essa função.

## 6. Referências

- [1] AL-DOGHMAN, Firas et al. A review on Fog Computing technology. In: Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on. IEEE, 2016. p. 001525-001530.
- [2] ALRAWAIS, Arwa et al. Fog Computing for the Internet of Things: Security and Privacy Issues. IEEE Internet Computing, v. 21, n. 2, p. 34-42, 2017.
- [3] ANTONI, Marco; VIVIAN, Gláucio Ricardo; PREUSS, Evandro. Implementação de uma nuvem de armazenamento privada usando Owncloud e Raspberry



- PI. Anais do EATI - Encontro Anual de Tecnologia da Informação e Semana Acadêmica de Tecnologia da Informação, 2015, Ano 5 n. 1, pp. 55-62.
- [4] BABU, Shaik Masthan; LAKSHMI, A. Jaya; RAO, B. Thirumala. A study on cloud based internet of things: Cloudiot. In: Communication Technologies (GCCT), 2015 Global Conference on. IEEE, 2015. p. 60-65.
- [5] BOTTA, Alessio et al. Integration of cloud computing and internet of things: a survey. *Future Generation Computer Systems*, v. 56, p. 684-700, 2016.
- [6] CHEN, Songqing; ZHANG, Tao; SHI, Weisong. Fog Computing. *IEEE Internet Computing*, v. 21, n. 2, p. 4-6, 2017.
- [7] CHIANG, Mung et al. Clarifying Fog Computing and Networking: 10 Questions and Answers. *IEEE Communications Magazine*, v. 55, n. 4, p. 18-20, 2017.
- [8] CRACIUNESCU, Razvan et al. Implementation of Fog computing for reliable E-health applications. In: *Signals, Systems and Computers*, 2015 49th Asilomar Conference on. IEEE, 2015. p. 459-463.
- [9] DÍAZ, Manuel; MARTÍN, Cristian; RUBIO, Bartolomé. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *Journal of Network and Computer Applications*, v. 67, p. 99-117, 2016.
- [10] FAN, Chih-Tien et al. Web Resource Cacheable Edge Device in Fog Computing. In: *Parallel and Distributed Computing (ISPD)*, 2016 15th International Symposium on. IEEE, 2016. p. 432-439.
- [11] HAJIBABA, Majid; GORGIN, Saeid. A review on modern distributed computing paradigms: Cloud computing, jungle computing and fog computing. *CIT. Journal of Computing and Information Technology*, v. 22, n. 2, p. 69-84, 2014.
- [12] HAO, Zijiang et al. Challenges and Software Architecture for Fog Computing. *IEEE Internet Computing*, v. 21, n. 2, p. 44-53, 2017.
- [13] HUO, Jiuyuan; QU, Hong; WU, Ling. Design and implementation of private cloud storage platform based on OpenStack. In: *Smart City/SocialCom/SustainCom (SmartCity)*, 2015 IEEE International Conference on. IEEE, 2015. p. 1098-1101.
- [14] JAIN, Akshay; SINGHAL, Priyank. Fog computing: Driving force behind the emergence of edge computing. In: *System Modeling & Advancement in Research Trends (SMART)*, International Conference. IEEE, 2016. p. 294-297.
- [15] KITCHENHAM, Barbara. Procedures for performing systematic reviews. Keele, UK, Keele University, v. 33, n. 2004, p. 1-26, 2004.
- [16] KUMAR, Praveen; ZAIDI, Nabeel; CHOUDHURY, Tanupriya. Fog computing: Common security issues and proposed countermeasures. In: *System Modeling & Advancement in Research Trends (SMART)*, International Conference. IEEE, 2016. p. 311-315.
- [17] LI, Songze; MADDAH-ALI, Mohammad Ali; AVESTIMEHR, A. Salman. Coding for Distributed Fog Computing. *IEEE Communications Magazine*, v. 55, n. 4, p. 34-40, 2017.
- [18] LINTHICUM, David S. Connecting Fog and Cloud Computing. *IEEE Cloud Computing*, v. 4, n. 2, p. 18-20, 2017.
- [19] LONGO, Francesco et al. Stack4things: An openstack-based framework for iot. In: *Future Internet of Things and Cloud (FiCloud)*, 2015 3rd International Conference on. IEEE, 2015. p. 204-211.
- [20] MARTINI, Ben; CHOO, Kim-Kwang Raymond. Cloud storage forensics: ownCloud as a case study. *Digital Investigation*, v. 10, n. 4, p. 287-299, 2013.
- [21] MCMILLIN, Bruce; ZHANG, Tao. Fog Computing for Smart Living. *Computer*, v. 50, n. 2, p. 5-5, 2017.
- [22] MRÓWCZYŃSKI, Piotr et al. Benchmarking and monitoring framework for interconnected file synchronization and sharing services. *Future Generation Computer Systems*, 2017.
- [23] NWOBODO, Ikechukwu (2015). A Comparison of Cloud Computing Platforms. *International Symposium on Circuits and Systems (ISCAS)*. Lisbon, Portugal, 24-27 May 2015. Atlantis Press.
- [24] OSANAIYE, Opeyemi et al. From cloud to fog computing: A review and a conceptual live VM migration framework. *IEEE Access*, 2017.
- [25] POPENTIU-VLADICESCU, Florin; ALBEANU, Grigore. Software reliability in the fog computing. In: *Innovations in Electrical Engineering and Computational Technologies (ICIEECT)*, 2017 International Conference on. IEEE, 2017. p. 1-4.
- [26] PRINCY, S. Emima; NIGEL, K. Gerard Joe. Implementation of cloud server for real time data storage using Raspberry Pi. In: *Green Engineering and Technologies (IC-GET)*, 2015 Online International Conference on. IEEE, 2015. p. 1-4.
- [27] STEINER, Wilfried; POLEDNA, Stefan. Fog computing as enabler for the Industrial Internet of Things. *e & i Elektrotechnik und Informationstechnik*, v. 133, n. 7, p. 310-314, 2016.
- [28] STOJMENOVIC, Ivan et al. An overview of Fog computing and its security issues. *Concurrency and Computation: Practice and Experience*, 2015.
- [29] VERBA, Nandor et al. Platform as a service gateway for the Fog of Things. *Advanced Engineering Informatics*, 2016.
- [30] ZHANG, Qi; CHENG, Lu; BOUTABA, Raouf. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, v. 1, n. 1, p. 7-18, 2010.
- [31] ZHU, Jiang et al. Improving web sites performance using edge servers in fog computing architecture. In: *Service Oriented System Engineering (SOSE)*, 2013 IEEE 7th International Symposium on. IEEE, 2013. p. 320-323.

**ARTIGO PUBLICADO**

**ARTIGO 2**

A Survey on Fog Computing and Its Research Challenges

**International Journal of Grid and Utility Computing – 2018**

# A Survey on Fog Computing and Its Research Challenges

José dos Santos Machado, Edward David Moreno, Admilson de Ribamar Lima Ribeiro

*Computer Department, Federal University of Sergipe – UFS*

jsmac18@hotmail.com, edwdavid@gmail.com, admilson@ufs.br

## Abstract

This paper reviews the new paradigm of distributed computing, which is Fog Computing, and it presents its concept, characteristics and areas of performance. It performs a literature review on the problem of its implementation and analyzes its research challenges such as: security issues; operational issues and their standardization. We show and discuss that many questions need to be researched in academia so that their implementation will become a reality, but it became clear that their adherence is inevitable for the internet of the future.

**Keywords:** Fog Computing, Cloud Computing, IoT, Distributed Computing, Cloud Integration to IoT, Research Challenges.

## 1. Introduction

In the last decades, it has been observed that computer services such as storage, data processing and control have been transferred to the "cloud". The opportunity for unlimited cloud computing allows final users to access extensive information easily, it is also possible to visualize that mobile devices and sensors, such as smartphones, have become powerful equipments, which has led to the emergence of new systems and applications.

These systems and applications introduce new functional demands on computing and networks that the "Cloud" alone cannot meet. In this scenario it is realized that local computing at a network edge is often necessary [5]. For example, to process real-time data, create location recognition contexts from local sensors, and maximize the efficiency of wireless communications at the edge of the network. However, the cloud is far away from devices to meet latency requirements and is very centralized to deal with heterogeneity and contextual diversity in a local area, it is also very costly to load all of the individual sensor data into the Cloud [26].

To overcome these limitations, portions of the cloud's computing capacity can be shifted to the edge of the network and form a local computing environment, that is, a "Fog Computing" also called "Fog". By distributing the closest computing and network services from which user data is generated, Fog is better suited to emerging demands [10].

In a Fog Computing architecture, sensor networks can play a significant role in the sense that sensors and actuators deployed in a distributed environment can become data generators, controllers for physical systems, and network computing platforms [31]. Fog

will have edge gateways that have more computing power. A network of sensors, including sensors and actuators, will be deeply connected to the gateways of the "Fog" and the mobile devices directly and then makes extensive interactions with them; this will provide new services that connect the physical environment to the cybernetic infrastructure [30].

Fog Computing can solve many Internet of Things (IoT) issues, for example, Fog services will be able to improve the bandwidth and cost constraints of long-range communications. However, many challenges remain in Fog Computing, how to model a system architecture for Fog; how to deploy, organize and manage Fog devices; how Fog interacts with the cloud and devices; and how to manage the physical and logical connectivity of Fog; among others.

This paper presents a review of the new paradigm of distributed computing that is the Fog Computing, its concept, characteristics and areas of performance are presented. It reviews the literature on the problem of its implementation and analyzes its research challenges.

The article is organized in five sections, section 2 presents the concept and characteristics of Fog Computing, section 3 is devoted to literature review, in section 4 we have the research challenges and finally in section 5 the conclusions and future work.

## 2. Fog Computing

This section presents and explains the concept, characteristics and areas of application and performance of the new paradigm of distributed computing, which is Fog Computing.

### 2.1 Concept of Fog Computing

Much of the work has focused on how to effectively offload device-intensive computing tasks with limited resources to the cloud and get the desired results [5].

However, because of the often unpredictable network latency, especially in a mobile environment, Cloud Computing can often fail to meet the stringent requirements for application latitude, security, and privacy in a geographically restricted area [28]. On the other hand, the increasing amount of data generated by devices and systems with few resources may become impractical to transport data through networks to remote Clouds [22].

For this, a new diagram appeared, Fog Computing. The Fog Computing concept was

adopted by Cisco Systems as a new paradigm in 2012 [10,30]. Fog Computing is Cloud Computing that will deliver advanced computing, storage, networking, and management services closer to end users by sending information from IoT devices for Cloud Computing, thus forming a distributed and virtualized platform, thus, it is also referred to as Edge Computing [13,31]. Figure 1 shows the location between Fog Computing and Cloud Computing.

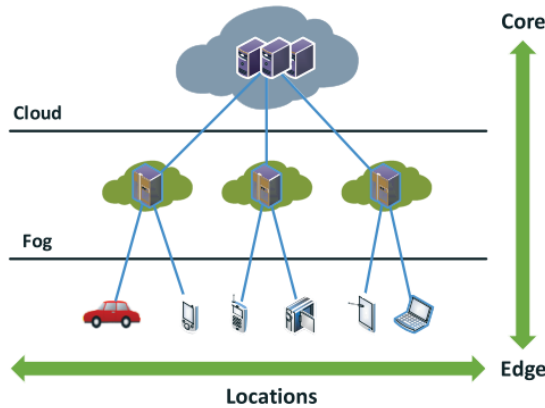


Fig. 1 - Fog located between edge and cloud [26]

Fog Computing offers many advantages desired by today's applications, such as real-time processing, fast scalability, and resource sharing and local storage. As such, Fog Computing quickly gained much attention from industry and academia. It is of course the bridge between the Internet of Things (IoT) and Cloud Computing, with the existing computing infrastructure on the Internet [15].

## 2.2 Characteristics of Fog Computing

Fog Computing is an innovative paradigm that performs distributed computing, network and storage services, and communication between Cloud Computing Data Centers to devices along the edge of the network [31]. This communication extends the operations and services inherent in Cloud Computing, thus allowing a new generation of applications [22]. The primary function is to filter and aggregate data for Cloud data centers and apply logical intelligence to final devices. Figure 2 presents Fog Computing architecture with its location.

Due to its recent introduction and emergence, there is no standard architecture available regarding Fog-based resource management, and for this reason there is still a simple model for this purpose, considering resource forecasting, resource allocation, and costs in a realistic and dynamic way, also considering the type and characteristics of the customers, this model is adaptable to the requirements of different customers.

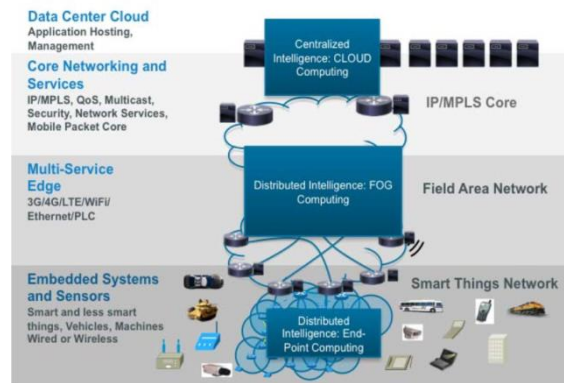


Fig. 2 – A Fog Computing Architecture [28]

The main features of Fog Computing are:

- **Heterogeneity:** A Fog Computing is a virtualized platform that delivers computing, networking and storage services between Cloud Computing and end devices of different types and shapes;
- **Geographic distribution:** Fog Computing has a widely distributed implementation to deliver high quality services for mobile and fixed end final devices;
- **Border location, location perception and low latency:** The concept of Fog Computing has been implemented to overcome lack of support for endpoints with quality services on the edge of the network;
- **Real-time interaction:** Various Fog applications, such as traffic monitoring systems, require real-time processing rather than batch processing;
- **Mobility support:** Mobility support is essential for many Fog Computing applications to enable direct communication with mobile devices using protocols such as the Cisco Location / ID Separation Protocol, which decouples the host identity from location identity using a distributed directory;
- **Large scale sensor networks:** This is applicable when monitoring the environment or network smart, using inherently distributed systems that require distributed computing or storage resources;
- **Prevalent for wireless access:** Most wireless access points and the mobile device gateway are typical examples of a Fog node in the network.
- **Interoperability:** Fog components must be capable of interoperating to ensure support for a wide range of services, such as data transmission.

Fog Computing is similar to Cloud Computing in many aspects, however, it can be differentiated from the previous one by the fact that it is close to the end devices, geographically smaller spatial distribution and the possibility of support for mobility [6]. As Fog-based processing occurs along the edge of the network, the final results reflect a highly improved perception of location, low latency and Quality of Service (QoS), in streaming and real-time

applications, Fog nodes are heterogeneous devices, ranging from servers, access points, edge routers to end devices such as cell phones, smart clocks, sensors and etc [1].

Fog works as a link between IoT and the Cloud to introduce the extra features needed for application-specific processing such as filtering and aggregation. Before transferring the data to the Cloud, you should be able to decide what to send (the content), how (data format) and when to send (time). During this process, it also needs to exclude some redundant or invalid data, and aggregate the complementary data in the space and time scaled [1]. Table 1 shows the comparison of results between the Cloud and the Fog.

Investigated Aspects	Traditional Cloud	Fog Computing
Latency prediction	5 seconds	1,5 seconds
Web Page Display Latency	8 seconds	3 seconds
Internet traffic	75 Kbps	10 Kbps
Used hardware	Amazon Web Server	Embedded systems, such as the Raspberry PI

Table 1 - Results between Cloud and Fog [1]

## 2.3 Areas of application and performance

Different applications of Fog Computing have been suggested in the literature. According to OSANAIYE et al. (2017) [22], the categories of Fog Computing applications are divided into: (i) Real-time applications; (ii) Almost real-time applications; (iii) Applications introduced in networks.

- (i) Real-time applications are low-latency applications that operate within a predefined time period and are classified by the user as immediate or urgent;
- (ii) Almost in real time, on the other hand, are applications that are subject to time delay introduced by data processing or network transmission, between the moment an event occurs and the use of the data for processing;
- (iii) Fog Computing can also be introduced into a network (for applications that do not require real-time processing and transmission) to reduce the amount of traffic in the processing core.

We will present some areas of application and performance of Fog Computing, available in the literature and that may benefit from its implementation.

### a) Video streaming

Broadcasting video applications and services will take advantage of some of the key benefits of Fog

Computing, including location knowledge, low latency, mobility support, and real-time analysis. Several smart devices support an intelligent surveillance application that can be used by law enforcement officers to view event streams in real-time suspect videos.

### b) Games

The advent of cloud computing provided a platform for non-user computer games (players) concerned with hardware requirements. Cloud gaming providers have recently expanded the Cloud infrastructure to deliver on-demand gaming to users over the Internet. It is offered remotely, allowing an interactive game that can be accessed and decoded by end devices such as smartphones or tablets.

### c) Health Care

IoT applications provided a structured approach to improving health services. This is achieved by deploying ubiquitous monitoring systems and transmitting data to Fog devices in real time before sending the information to the Cloud for further analysis and diagnosis.

### d) Intelligent Traffic light System (ITLS)

The intelligent traffic light has been idealized by intelligent vehicles connected and advanced transportation system. It interacts locally with a series of sensor nodes to detect the presence of cyclists, motorcyclists or pedestrians and also estimates the speed and distance of the approaching vehicles. This information can be used to prevent accidents by sending warning signals to approaching vehicles.

### e) Smart Cities

A smart city is the essential key to IoT application, which ranges from intelligent traffic management to building energy management. The concept of an intelligent city has attracted the great interest of the science, engineering, research community and professional sectors as a means to overcome the challenges associated with rapid urban growth.

### f) Intelligent Vehicles

The advent of mobile Cloud Computing has required the study of its agents, such as vehicles, robots, and humans, who interact together to detect the environment, process data, and transmit results. Vehicle connected through Fog Computing can communicate with its internal and external environment.

### g) Smart Grid

Power load balancing applications can run on network edge devices, such as smart meters and micro grids. Based on the energy demand and the possibility of a lower cost, these devices

automatically switch to alternative energies such as solar energy and wind.

#### ***h) Sensor Networks and Wireless Actuators***

Traditional wireless sensor networks are insufficient in applications that go beyond detection and tracking, which require actuators to perform physical actions such as opening, closing or even moving sensors. In this scenario, actuators that act as Fog devices can control the measurement process itself, stability, and oscillatory behaviors by creating a closed loop system.

#### ***i) Intelligent Building Control***

The applications of this scenario are facilitated by wireless sensors implanted to measure temperature, humidity or levels of various gases in the building atmosphere. In this case, information can be exchanged between all sensors on a floor, and their readings can be combined to form reliable measurements. The sensors will use decision making and the activation of Fog devices to react with incoming data information.

#### ***j) IoT and Cyber Physical Systems (ICSs)***

Fog Computing-based systems are becoming an important class of IoTs and ICSs. Based on traditional information carriers, including the Internet and telecommunications network, IoT is a network that can interconnect common physical objects with identified addresses. The ICSs present a combination of the computational and physical elements of the system.

#### ***k) Software Defined Networks (SDN)***

The Fog Computing framework can be applied to implement the SDN concept for vehicle networks. SDN is an emerging paradigm of network computing and has become one of the most popular topics in the IT industry. It separates layers of control and data communication. The control is performed on a centralized server and the nodes follow the communication path decided by the server.

#### ***l) Self-sustaining train***

Train maintenance is an important area of application. Temperature variations can be detected by the ball bearing monitoring sensor on the train and can automatically notify any type of disorder. This will help in disaster prevention.

Current and future applications that require Fog Computing include connected vehicles, autopilot vehicles, smart grids, sensor networks and wireless actuators, smart homes, smart cities, connected oil and gas systems, and mobile health systems [5].

### **3. Literature revision**

Cloud Computing is a form of Internet-based computing that can provide hardware and software capabilities as needed [30]. Cloud Computing virtualizes the computing, networking, storage, and other infrastructure to form a wide range of shared, measurable and dynamic resources, and provides a computing model for all types of users in the form of services controlled through a management platform [3,4,8,12].

The Internet of Things (IoT) are the smart connected devices that have grown exponentially in terms of technologies, market share and consumer approval, paving the way for the evolution of the Fog Computing principles, gradually improving productive opportunities in various domains such as vehicle networks and Smart Grid, the advantages of this computing procedure for cross-domain services are needed and should be investigated.

Fog Computing enables greater support and improved Internet response time of Things, is suitable for real-time service requests, and to share resources efficiently, an efficient and cooperative pairing strategy-based utility is required between the IoT nodes [25,30].

#### **3.1 Systematic review**

In order to reach a degree of scientific rigor, we sought to ensure the research process from the concepts of Systematic Review (SR) as described below.

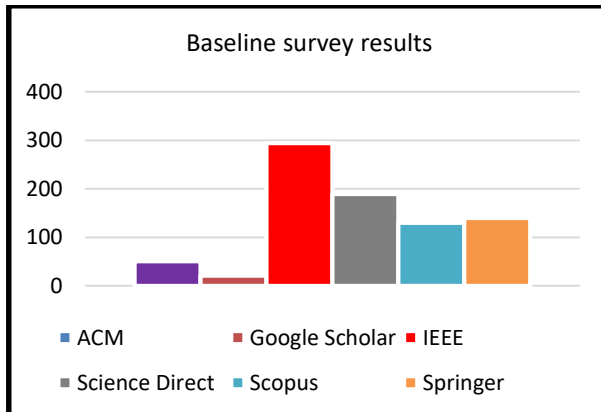
The systematic review of literature identifies, evaluates and interprets all available research relevant to a specific issue, thematic area, or phenomenon of interest.

The importance of establishing a systematic review process is fulfilled when observing the definition of phases, which are: a) Review planning; b) Procedures for conducting and extracting SR, and, finally, c) SR reporting procedures.

Thus, the results are more reliable in relation to the revision of the primary literature, due to its rigorous form and that gives possibility of repetitions and auditing.

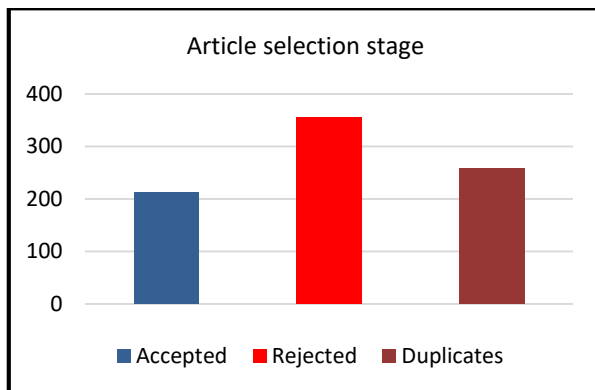
Thus, RS focused on the characterization of the solution development processes for the implementation of Fog Computing.

A total of 825 articles (graphic1) were identified, with a total of 51 articles, GOOGLE SCHOLAR (21 articles), IEEE (294 articles), SCIENCE DIRECT (189 articles), SCOPUS (130 articles) and SPRINGER (140 articles).



Graph 1 - Articles identified on Fog Computing  
Source: Own by the author (2017)

At that time, the criteria for exclusion of the 825 papers related to the identification process were evaluated, and 212 papers were accepted to be evaluated in the extraction phase of the literature review, 613 of which were excluded, 355 of which were defined by the exclusion protocol which did not present a complete text, works where the search terms do not appear in the fields: title, abstract and keywords and course completion papers) and 258 excluded works (graphic 2) because they are duplicated when exporting the results of queries of the bases.



Graph 2 - Result of the selection stage of the articles  
Source: Own by the author (2017)

In order to establish the interests of the research, the protocol constructed defined the following selection criteria: articles published in the period from 2013 to 2017; complete articles available online; English and Portuguese language; studies that address areas of Fog Computing and studies that address the integration of the Cloud with IoT.

The 212 papers accepted by at least one selection criterion and were taken to the extraction stage for the complete reading process, where the conclusion of 33 papers was completed, which met the criteria formalized in the protocol of the systematic review.

### 3.2 Analyzed works

A total of nine articles were identified in the implementation of Fog Computing to improve some of the integration services between the Cloud and IoT devices, we will present their results and their future research work.

ZHU et al. (2013) [28], presented web optimization within the Fog Computing context. They have applied existing methods for web optimization in an innovative way, in such a way that these methods can be combined with unique knowledge that is only available on edge (*Fog*) nodes. As future work they signaled their proposed concepts to develop a proof-of-concept system.

In the work of CRACIUNESCU et al. (2015) [7], they presented an e-Health laboratory implementation, where real-time processing is performed by the home PC, while the extracted metadata is sent to the Cloud for further processing. As future work signaled add more sensors and more off-the-shelf devices, which are currently mainstream, and the fusion data of these devices.

In [27] VERBA et al. (2016), they analyzed the existing platforms and their shortcomings, as well as to propose a modular platform, based on messages that allows the grouping of gateways and the abstraction of the details of the protocol of peripheral communication. And they flagged as future work developing an intelligent lab environment with several more complex devices and control scenarios to fully test the system.

FAN et al. (2016) [9], introduced the ability of the Web Caching feature added to the edge device to serve as a caching proxy server for more caching. End devices are also exploited to provide some cache space. As future work they signaled adding more functionality to the edge device, such as security and implementing work in the real world.

In the work of HAO et al. (2017) [11], they presented a WM-FOG design, a computing framework for Fog environments, which encompasses this software architecture and evaluates its prototype of the system. As future work signaled add more features to WM-FOG to better serve Fog Computing applications.

MANOGARAN et al. (2017) [33], proposes a new architecture for the implementation of IoT to store and process scalable sensor data (big data) for health care applications. Architecture consists of two main sub architectures, namely, Meta Fog-Redirection (MF-R) and Grouping and Choosing (GC) architecture. MF-R architecture uses big data technologies such as Apache Pig and Apache HBase for collection and storage of the sensor data (big

data) generated from different sensor devices. The proposed GC architecture is used for securing integration of Fog Computing with cloud computing. This architecture also uses key management service and data categorization function (Sensitive, Critical and Normal) for providing security services. The framework also uses MapReduce based prediction model to predict the heart diseases. Performance evaluation parameters such as throughput, sensitivity, accuracy, and f-measure are calculated to prove the efficiency of the proposed architecture as well as the prediction model.

In [15] LI et al. (2017), discussed two recently proposed encoding concepts, minimum bandwidth codes and minimal latency codes, and illustrate their impacts on Fog Computing, also discussed a unified coding framework that includes the two coding techniques described above. As future work signaled the need to research heterogeneous computing; networks with multiple layer and structured topology; multi-step computing tasks; coded computing costs; verifying distributed computing; explore the algebraic structures of computational tasks; heavy-duty communication applications and Fogplug-and-play nodes.

OSANAIYE et al. (2017) [22], presented a conceptual real-time migration approach for migration of VMs and signaled as future work the implementation of the framework in the real world or test environment, with the purpose of its validation.

Finally, POPENTIU-VLADICESCU et al. (2017) [23], analyzed the existing models, architectures and practices in Fog computing for the reliability and safety of Fog systems, an approach considered integrating three components of system reliability: nodes reliability, network reliability and software reliability, OpenFog reference architecture, and AJIA and BDSC schemas. As future work signaled solving technical problems and algorithms in the Fog Computing paradigm.

The analysis of the presented studies allowed to identify that, the scientific production on the implementation of Fog Computing is still scarce. Therefore, it is necessary to invest more in research in this area, given the importance of implementing Fog Computing as a technology that will promote security and speed in data transmission between the Cloud and IoT.

In Table 2 the data of the articles analyzed are summarized and presented, were organized in increasing order chronologically to demonstrate the evolution in relation to the theme Fog Computing. The articles were compared regarding the implementation of Fog Computing, the use of Cloud platform, use of embedded device and the use of techniques or methods of evaluation (Benchmark).

Paper	Fog Computing	Platform Cloud	Embedded device	Benchmark (Evaluation)
[28] ZHU (2013)	✓			
[7] CRACIUNESCU (2015)	✓	✓	✓	
[27] VERBA (2016)	✓	✓	✓	✓
[9] FAN (2016)	✓			✓
[11] HAO (2017)	✓	✓		✓
[33] MANOGARAN (2017)	✓	✓		✓
[15] LI (2017)	✓			✓
[22] OSANAIYE (2017)	✓			
[23] POPENTIU-VLADICESCU (2017)	✓			

Table 2 - Comparison between works  
Source: Own by the author (2018)

## 4. Research Challenges

Fog Computing is an extension of the classic Cloud Computing to the edge of the network (as the mist is a cloud near the ground). It is designed to support Internet Applications of Things (IoT) characterized by limitations of latency, mobility requirement and distribution geographical area. Although computing, storage and networking are both Cloud and Fog features, the latter has specific features, edge location and site recognition implying low latency, geographic distribution, and a large number of nodes in contrast to the centralized Cloud, support for mobility (via wireless access) and real-time interaction (instead of batch processes), support for interaction with the Cloud [9,13].

In [32] LIN et al. (2017), presented a comprehensive overview of IoT with respect to system architecture, enabling technologies, security and privacy issues, and present the integration of fog / edge computing and IoT, and applications. Explores the relationship between cyber-physical systems and IoT, existing architectures, enabling technologies, and security and privacy issues in IoT. Investigate the relationship between IoT and fog / edge computing, and discuss issues in fog / edge computing-based IoT.

In [4] the authors have done an analysis that shows how the construction of Fog Computing projects is a challenge, in fact, the adoption of Fog-based approaches requires several algorithms and methodologies related to the reliability of smart device networks, and that operate under specific conditions calling for specific fault tolerance techniques.

MAHMUD et al. (2018) [31], presented a taxonomy of Fog computing according to the identified challenges and its key features. Also maps the existing works to the taxonomy in order to identify current research gaps in the area of Fog computing. Moreover, based on the observations, proposes future directions for research.



The Fog Computing is proposed to solve the problems mentioned above. As Fog Computing is implemented at the edge of the network, it provides low latency, geographical location definition, and improves quality of service (QoS) for applications that use streaming and real-time applications [14]. Typical examples include industrial automation, transportation, sensor networks and actuators [30]. In addition, this new infrastructure supports heterogeneity, as Fog devices include devices for end users, access points, routers and switches. As stated in [26], the Fog paradigm is well-positioned for real-time data analysis, supports densely distributed data collection points, and offers advantages in entertainment, advertising, personal computing, and other applications.

## 4.1 Security Issues

As a "small Cloud" near end users, Fog devices may face system security challenges and should be deployed in strict surveillance and protection locations [32]. In order to protect against malicious purposes such as espionage and data hijacking. Traditional attacks become available to compromise the Fog device system [26].

LI, Xiong et al. (2017) [29], proposes a secure remote authentication scheme employing the chaotic maps. Use the formal proof under random oracle model, and the famous verification tool Proverif to prove the security of the proposed scheme. Besides, informal analysis including ten security properties and performance comparison are shown to supplement the security properties. Which can be searched for use in Fog Computing. In this section, we discuss the system security issues in Fog Computing.

### a) Intrusion detection techniques

Intrusion detection techniques can also be applied in Fog Computing. Intrusion into intelligent networks can be detected using a signature-based method in which behavior patterns are observed and verified in a database. Intrusion can also be captured using an anomaly-based method in which an observed behavior is compared to the expected behavior to check for a misdirection.

In [26], the authors present an example of a man attack in the middle. The man-in-the-middle attack has the potential to become a typical attack on Fog Computing. In this attack, the gateways that serve as Fog devices can be compromised or replaced with false ones. Examples are KFC or Star Bar clients that connect to malicious access points that provide misleading service set identifiers as legitimate. The private communication of the victims will be kidnapped once the attackers take control of the gateways.

### b) Authentication and authorization

Authentication and authorization issues have not been studied in the context of Fog Computing. They were studied in the context of intelligent networks and machine-to-machine communication [26]. There are security solutions for Cloud Computing. However, they may not suit Fog Computing, because Fog devices run at the edge of networks. The Fog devices work environment will face many threats that do not exist in a well-managed Cloud.

Fog devices typically have some kind of connectivity to the Cloud Remote Authentication Server, which can be used to distribute authentication information and collect audit logs, but this connectivity may be very slow in certain environments.

In addition, dependency on cloud central authentication servers is not ideal because authentication should continue to be applied to the person accessing the devices locally when the remote authentication server communication is disabled. A provision to ensure that necessary access is available in emergency situations can be important, even if it means ignoring normal access control, but with a certain audit control.

MANOGARAN et al. (2017) [30], proposes an efficient centralized secure architecture for end-to-end integration of IoT based healthcare system deployed in cloud environment. The proposed platform user Fog Computing environment to run the framework. The main focus of this work is to secure Authentication and Authorization of all the devices. The proposed system uses asynchronous communication between the applications and data server deployed in the cloud environment.

### c) Network security

The development of wireless technology has led to increased security problems. Fog Computing is affected in similar ways, like other wireless technologies. Several examples of such attacks are sniffer, spoofing, jamming, etc. These attacks usually occur between the Fog node and the centralized Cloud system. Usually, you rely on the settings defined by the central network system, which separates normal data traffic from the client network. For this reason it greatly overloads the network manager. In addition, the mist nodes are at the edge of the network, which increases the handling of the network manager. SDN (Software Defined Networking) can be used as an approach for the network manager to work at the low level of abstraction for network services [14]. It can help manage, increase network scalability, and reduce cost with respect to Fog Computing.

### d) Data security

In Fog Computing the user control for the data is transmitted by the Fog node, hence the same security issues arise from Cloud Computing. The integrity of

the data can't be maintained when the data is lost or modified. Data that is loaded into the Fog node can also be used by third parties. There are several techniques that can be used to provide data integrity, confidentiality and veracity, such as the combination of homomorphic encryption and searchable cryptography [14]. These techniques ensure that the client does not store data on the untrusted server. Also the data can be retrieved much more quickly and therefore the cost of communication becomes low. There will always be new challenges in Fog Computing related to the storage system that can handle the dynamic operations much faster and take up less space.

#### e) Privacy

User privacy is paramount in any computer system, this issue is of great concern also in Fog Computing. Not only the service provider, but the fact that the government is also involved makes it more challenging. In fact, it becomes even easier for the third party to have access to private user data, location, and so on. And therefore, the user becomes prone to attacks. Due to the delivery of data at the edge of the Fog node, it becomes much easier to collect all the important information from the users. This is one of the most challenging problems in Fog Computing, preserving the privacy of users [14].

#### f) Safe and efficient protocols

Many existing protocols are time-based synchronization and packet transmissions over wireless networks and are not suitable for IoT devices with limited capabilities in Fog Computing. Wireless transmissions and safety calculations consume a significant fraction of the power capacity [2]. The main challenge is how to design safe and effective schemes in IoT without sacrificing performance and consuming a high energy rate.

#### g) Location verification

In hostile environments such as vehicles and railways, IoT devices in a Fog environment can move quickly and dynamically, making it difficult to verify location. The main challenge is how to design a secure and accurate location verification scheme in such volatile environments and at the same time, the scheme should be suitable for IoT devices with limited resources.

### 4.2 Operational Issues

In [11], the authors present a series of questions about the operation of Fog Computing, as questions to be researched and defined.

- (i) For Fog nodes with heterogeneous hardware, is it acceptable to switch energy efficiency for reduced latency?

- (ii) Should a process on a Fog node be interrupted when the user moves toward another Fog node?
- (iii) How should tasks be scheduled considering latency, power consumption, mobility, and existing workload?
- (iv) Where should the scheduling program be implemented?
- (v) What are the benefits of joint scheduling tasks?
- (vi) When should services be started and discontinued?
- (vii) What is the best way to balance the workload?
- (viii) Should the virtualization to be used be VM-based or container-based?
- (ix) What methodologies should be used to efficiently provision services for hundreds, thousands, and millions of users?
- (x) What is the best way to split the workload for services that aggregate information from nearby customer devices to client-side energy efficiency?

In Cloud Computing, data consistency can be achieved by coordinating the Cloud Servers in the data centers where the Cloud is deployed. In Fog Computing, however, things get complicated. When deploying data objects in a Fog environment, it is necessary not only to coordinate the back-end Cloud Servers, but also to validate the cached data on the Fog nodes as well as on the client devices, if a strong consistency of data [11]. This can result in deteriorated write performance, which weakens the benefits of using Fog nodes as the write cache servers. On the other hand, Fog Computing also offers opportunities to achieve data consistency more efficiently than Cloud Computing.

### 4.3 Standardization

Cisco Systems tries to standardize on Fog Computing and has created OpenFog Consortium to promote its vision. Whatever is called and defined, the key is to reduce latency for critical applications in latency by moving data transfer and processing to the edge of the network, closer to the IoT device [16].

It can be applied across dozens of systems where data and applications have been deployed near the edge of the network, but still work with centralized data and applications. Although it's a bit complicated, it's not that hard to do. OpenFog will benefit on many factors by adhering to Fog Computing, such as:

A standard architecture and enabling technology that allows you to approach Fog Computing in a simple but consistent way;

The ability to provide a good product development framework that network devices and softwarebuilders can follow, as well as influence their development;

The ability to deal with safety consistently. DDOS attacks have taken over devices, computers, and now that everything is smart and has an operating system, this will be a critical factor in the internet of the future.

OpenFog recently published a reference architecture to standardize the implementation of Fog Computing, based on eight pillars to encompass a standard architecture considering aspects of security, scalability, openness, autonomous, partner, agile, hierarchical and programmability (see figure 3).



Fig. 3 OpenFog reference architecture [16]

Open architecture is needed to develop an intelligent living environment with potentially thousands of vendors [19]. The OpenFog consortium is comprised of EDGE Lab companies from Cisco, Dell, Intel, Microsoft and Princeton to accelerate the adoption of Fog Computing ([www.openfogconsortium.org](http://www.openfogconsortium.org)) and is taking steps to create a common reference architecture that covers hardware platform and software with highly sophisticated features.

## 5. Conclusion and future work

Fog Computing will be able to deliver advanced computing, storage, network and management services closer to end users along the edge of the network, between the Cloud, IoT and mobile devices, thus forming a distributed and virtualized platform.

Problems of latency, storage and processing power of these devices have made Fog Computing appear. Thus, Fog Computing quickly attracted much attention from industry and academia.

This work carried out a review of the new paradigm of distributed computing that is Fog Computing, presented its concept, characteristics and areas of action. Here we have reviewed the literature on the problem of its implementation and analyzed its research challenges such as: security issues; operational issues and their standardization.

We have showed that many questions need to be researched in academia so that their implementation is a reality, but it became clear that their adherence is inevitable for the internet of the future.

For further work we intend to propose the implementation and analysis of a Fog Computing, to provide StaaS (Storage as a Service), to IoT devices using an embedded platform based on Raspberry PI.

It can be a good alternative to solve problems related to the storage of IoT data, serving as Fog Computing to store data of these devices and send to Cloud Computing, safely and efficiently in the transport of data between networks. Using embedded systems on a low-cost device platform, instead of using powerful and expensive servers to perform this function.

## 6. References

- [1] AL-DOGHMAN, Firas et al. A review on Fog Computing technology. In: Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on. IEEE, 2016. p. 001525-001530.
- [2] ALRAWAIS, Arwa et al. Fog Computing for the Internet of Things: Security and Privacy Issues. IEEE Internet Computing, v. 21, n. 2, p. 34-42, 2017.
- [3] BABU, ShaikMasthan; LAKSHMI, A. Jaya; RAO, B. Thirumala. A study on cloud based internet of things: Cloudiot. In: Communication Technologies (GCCT), 2015 Global Conference on. IEEE, 2015. p. 60-65.
- [4] BOTTA, Alessio et al. Integration of cloud computing and internet of things: a survey. Future Generation Computer Systems, v. 56, p. 684-700, 2016.
- [5] CHEN, Songqing; ZHANG, Tao; SHI, Weisong. Fog Computing. IEEE Internet Computing, v. 21, n. 2, p. 4-6, 2017.
- [6] CHIANG, Mung et al. Clarifying Fog Computing and Networking: 10 Questions and Answers. IEEE Communications Magazine, v. 55, n. 4, p. 18-20, 2017.
- [7] CRACIUNESCU, Razvan et al. Implementation of Fog computing for reliable E-health applications. In: Signals, Systems and Computers, 2015 49th Asilomar Conference on. IEEE, 2015. p. 459-463.
- [8] DÍAZ, Manuel; MARTÍN, Cristian; RUBIO, Bartolomé. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. Journal of Network and Computer Applications, v. 67, p. 99-117, 2016.
- [9] FAN, Chih-Tien et al. Web Resource Cacheable Edge Device in Fog Computing. In: Parallel and Distributed Computing (ISPD), 2016 15th International Symposium on. IEEE, 2016. p. 432-439.
- [10] HAJIBABA, Majid; GORGIN, Saeid. A review on modern distributed computing paradigms: Cloud computing, jungle computing and fog computing. CIT. Journal of Computing and Information Technology, v. 22, n. 2, p. 69-84, 2014.
- [11] HAO, Zijiang et al. Challenges and Software Architecture for Fog Computing. IEEE Internet Computing, v. 21, n. 2, p. 44-53, 2017.
- [12] HUO, Jiuyuan; QU, Hong; WU, Ling. Design and implementation of private cloud storage platform based on OpenStack. In: Smart City/SocialCom/SustainCom (SmartCity), 2015 IEEE International Conference on. IEEE, 2015. p. 1098-1101.
- [13] JAIN, Akshay; SINGHAL, Priyank. Fog computing: Driving force behind the emergence of edge computing. In: System Modeling & Advancement in Research Trends (SMART), International Conference. IEEE, 2016. p. 294-297.
- [14] KUMAR, Praveen; ZAIDI, Nabeel; CHOUDHURY, Tanupriya. Fog computing: Common security issues and proposed countermeasures. In: System Modeling &

- Advancement in Research Trends (SMART), International Conference. IEEE, 2016. p. 311-315.
- [15] LI, Songze; MADDAH-ALI, Mohammad Ali; AVESTIMEHR, A. Salman. Coding for Distributed Fog Computing. *IEEE Communications Magazine*, v. 55, n. 4, p. 34-40, 2017.
- [16] LINTHICUM, David S. Connecting Fog and Cloud Computing. *IEEE Cloud Computing*, v. 4, n. 2, p. 18-20, 2017.
- [17] LONGO, Francesco et al. Stack4things: An open stack-based framework for IoT. In: *Future Internet of Things and Cloud (FiCloud)*, 2015 3rd International Conference. IEEE, 2015. p. 204-211.
- [18] MARTINI, Ben; CHOO, Kim-Kwang Raymond. Cloud storage forensics: ownCloud as a case study. *Digital Investigation*, v. 10, n. 4, p. 287-299, 2013.
- [19] MCMILLIN, Bruce; ZHANG, Tao. Fog Computing for Smart Living. *Computer*, v. 50, n. 2, p. 5-5, 2017.
- [20] MRÓWCZYŃSKI, Piotr et al. Benchmarking and monitoring framework for interconnected file synchronization and sharing services. *Future Generation Computer Systems*, 2017.
- [21] NWOBODO, Ikechukwu (2015). A Comparison of Cloud Computing Platforms. *International Symposium on Circuits and Systems (ISCAS)*. Lisbon, Portugal, 24-27 May 2015. Atlantis Press.
- [22] OSANAIYE, Opeyemi et al. From cloud to fog computing: A review and a conceptual live VM migration framework. *IEEE Access*, 2017.
- [23] POPENTIU-VLADICESCU, Florin; ALBEANU, Grigore. Software reliability in the fog computing. In: *Innovations in Electrical Engineering and Computational Technologies (ICIEECT)*, 2017 International Conference. IEEE, 2017. p. 1-4.
- [24] PRINCY, S. Emima; NIGEL, K. Gerard Joe. Implementation of cloud server for real time data storage using RaspberryPi. In: *Green Engineering and Technologies (IC-GET)*, 2015 Online International Conference. IEEE, 2015. p. 1-4.
- [25] STEINER, Wilfried; POLEDNA, Stefan. Fog computing as enabler for the Industrial Internet of Things. *e& i Elektrotechnik und Informations technik*, v. 133, n. 7, p. 310-314, 2016.
- [26] STOJMENOVIC, Ivan et al. An overview of Fog computing and its security issues. *Concurrency and Computation: Practice and Experience*, 2015.
- [27] VERBA, Nandor et al. Platform as a service gateway for the Fog of Things. *Advanced Engineering Informatics*, 2016.
- [28] ZHU, Jiang et al. Improving web sites performance using edge servers in fog computing architecture. In: *Service Oriented System Engineering (SOSE)*, 2013 IEEE 7th International Symposium on. IEEE, 2013. p. 320-323.
- [29] LI, Xiong et al. A secure chaotic map-based remote authentication scheme for telecare medicine information systems. *Future Generation Computer Systems*, 2017.
- [30] MANOGARAN, Chandu Thota Gunasekaran; PRIYAN, M. Centralized fog computing security platform for IoT and cloud in healthcare system. *Exploring the convergence of big data and the internet of things*, p. 141, 2017.
- [31] MAHMUD, Redowan; KOTAGIRI, Ramamohanarao; BUYYA, Rajkumar. Fog computing: A taxonomy, survey and future directions. In: *Internet of Everything*. Springer, Singapore, 2018. p. 103-130.
- [32] LIN, Jie et al. A survey on internet of things: architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal*, 2017.
- [33] MANOGARAN, Gunasekaran et al. A new architecture of Internet of Things and big data ecosystem for secured smart healthcare monitoring and alerting system. *Future Generation Computer Systems*, 2017.